EECS 461 Problem Set #2
Due Date: Tuesday, December 8, 5pm

This problem set will be weighted one third of the total number of points you missed (or will miss) on the midterm and final, or roughly 1% per question. The intent is that the problem set will be optional to those people who did well on the midterm, and strongly encouraged for those students who scored lower than the mean. For people in the middle, our advice is to focus on the problems you believe you will get the most out of doing, to maximize understanding per time spent.

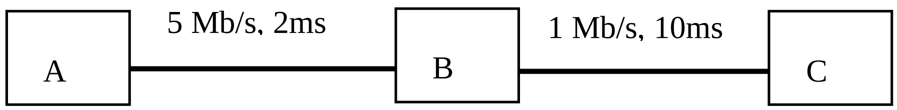
We've set the due date to give you maximum flexibility, but that puts an added burden on you to allocate your time wisely. We advise reading over the problems and letting them simmer, instead of trying to do them all at once.

Since we won't be able to grade the problem sets in time for the final, we will release a solution set immediately after the due date. Please keep a copy to check your own answers.

Questions are divided into "easier" and "harder" sets; these are only relative terms -- you may find some questions in the easier set hard to answer, and others in the harder set easy to answer. It is meant solely as a rough guide. Questions in the easier set are like those that might appear on an exam (several are taken from last year's final). All questions are weighted equally. You are free to discuss this assignment with other students and the TA, but you must each individually write up and submit your own solutions.

Easier Questions

1. Rate/Delay Concept. Suppose three computers (A, B, and C) are interconnected by two links as follows:



a. How long does it take for A to send an 8KB packet to B and for B to fully receive that packet?

Time to send packet = propagation delay + bandwidth delay
= 2ms + 8KBs/5Mb
= 14.8 ms.

b. How long does it take for A to send an 8KB packet to C (via B) and for C to fully receive that packet, assuming that B must receive the entire packet before forwarding it to C (the last bit from A has to arrive at B before B can send the first bit to C -- store and forward)?

Time to send from A to B = 14.8 ms
Time to send from B to C = 10ms + 8KBs/1Mb
= 74 ms
Total time = 88.8 ms.

c. How long does it take for A to send an 8KB packet to C (via B) and for C to fully receive that packet, assuming the 8KB packet was sent as 16 separate 512 byte chunks, again assuming B is store and forward?

Suppose, A starts the transmission of the packet at time 0.
Then the first chunk of data completely reaches B at time t = 2ms + 0.5KBs/5Mb
= 2.8ms
The *i*th chunk of data completely reaches B at time t_i = 2+0.8*i ms

B starts sending the data as soon as it receives the first chunk of data completely.
B is ready to send the i*th* chunk of data at time = 2.8 + 0.5KBs/1Mb * [i – 1]
= 2.8 + 4 * [i –1] ms
Note that each chunk of data arrives much before B is ready to send it.

The last chunk of data reaches C at time = time of starting to send + delay
= (2.8 + 4 * 15) + 4 + 10
= 76.8 ms

d. Assume now that A sends packets to C continuously (i.e., back to back) at a rate dictated by the link between A and B. Assume B can queue at most 10 packets at any time. How long after A starts blasting packets will B be forced to drop a packet? Beyond that point, how often will B drop packets? If B's queue size is increased to 20 packets, how often will it drop packets (assuming A still sends continuously)?

Assume a packet size of 0.5KB.
B receives a packet after every 0.8ms.
B can send one packet in 4ms.
After 8ms, B has received 10 packets while sent only 2. Therefore its queue length is 8.
After another 1.6ms, it has received 2 more packets but is still in the process of sending the third packet.
Assuming that the packet being sent is in the queue, the time after which the first packet would be dropped is 2ms (initial propagation delay) + 9.6ms = 11.6ms
Thereafter, B is going to drop 4 out of every 5 packets.

If the queue size is increased, the dropping rate remains the same, although the first drop is delayed.

e. Now, instead, assume A sends packets back-to-back until it receives an acknowledgment packet from C, at which time it immediately stops sending. Assume that C generates this acknowledgment as soon as it receives the last bit of the first packet from A. How many packets will A send? Does the queue at B overflow?

C receives the first packet completely at time t = 16.8ms
Assuming that the ACK is the same size as the data packet, A completely receives the ACK at time 33.6ms
Packets sent by A during that time = 42. Yes, the queue overflows during that time.

2. Derive a formula for the efficiency of a stop and wait protocol, assuming a full duplex link, in terms of: T (the timeout value), p (the probability that the link drops a packet), d (the propagation delay, i.e., the transmission time that would be experienced by a 1 bit packet), r (bandwidth of the link), s (packet size), and h (header size). Hint: it is useful to define intermediate terms, e.g., the round trip time for a packet of size s, etc.

Time to transmit a data packet from sender to receiver = s/r + l
Time to transmit ACK (assuming ACK size = h) = h/r +l

Round trip time = $2l + \frac{h}{r} + \frac{s}{r}$
Probability that neither the data nor the ACK is dropped = q = (1-p)*(1-p)
Probability that at least one of them is dropped = 1-q
Average number of re-transmissions of a packet = $0*q + 1(1-q)q+2(1-q)^2q+\ldots$
= (1-q)/q
Expected time for sending each packet T_{send} **= T(1-q)/q + RTT**

$$= T\frac{1-q}{q} + 2l + \frac{h}{r} + \frac{s}{r}$$

Efficiency = $s/(T_{send}*r)$ **=** $\frac{s}{Tr(1-q)/q + 2lr + h + s}$

3. Write a pair of RPC stubs to turn your code for homework #1 into a "Microsoft stock quote server". In other words, the client calls "getQuote()", but the actual code to implement the routine runs on a different machine. For extra credit, demonstrate that your code works by running it on two separate machines.

The following is a high level description for the client and server side stubs. Assume that the code to implement the getQuote method is on machine A.

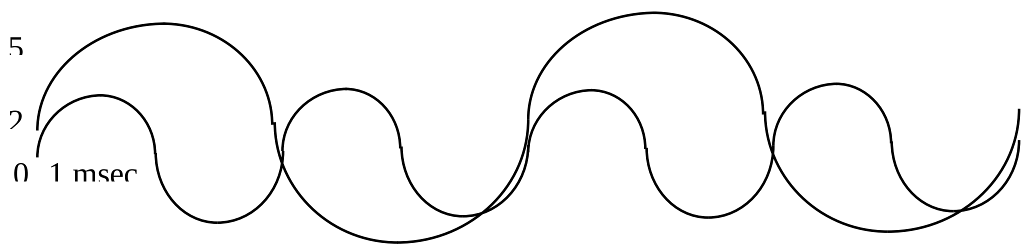
<u>Client Stub:</u>
Establish a TCP connection with A
Send a message to A with the contents "getQuotes"
Wait for the reply.
On getting a reply from A, remove a floating point number from the reply message.

<u>Server Stub:</u>
Wait for a client to establish connection
Spawn a new thread to handle the client request.
Parse the client request to determine the method to be called.
Call the method (For instance, if the client requested the method "getQuotes", then call the local method "getQuotes()"
Bundle the result in a message and send the message back to the client

4. The book claims that the initial sequence numbers (ISN) of a TCP connection are chosen "randomly" by each end point. This is not quite true. Find out what TCP actually does to choose ISNs and briefly describe the algorithm. You might look this up in the TCP specification itself, as described in "Request for Comments 793" (RFC-793). To find an on-line copy, try your favorite search engine or go to http://www.ietf.org/, and follow links.

The initial sequence number is generated using a 32 bit clock whose low order bit is incremented roughly every 4 microseconds.

5. The Fourier analysis of an input signal has already been done for you and shown to consist of two sine waves, one with period 2 milliseconds and amplitude 2, and one with period 4 milliseconds and amplitude 5. Suppose the frequency response of the medium scales frequencies at or below 250 Hz by a factor of 1 (they come out as they go in), and frequencies at or above 500 Hz by a factor of 2 (they come out twice as big as how they come in). Reconstruct the input and output signals by constructing a table listing the values of each of the signals at 250 microsecond intervals. Use this table to sketch the input and output signals.



Since the first signal has a frequency of 500Hz, it is attenuated by a factor of 2. The second signal remains the same. The actual input and output are the sum of the 2 signals.

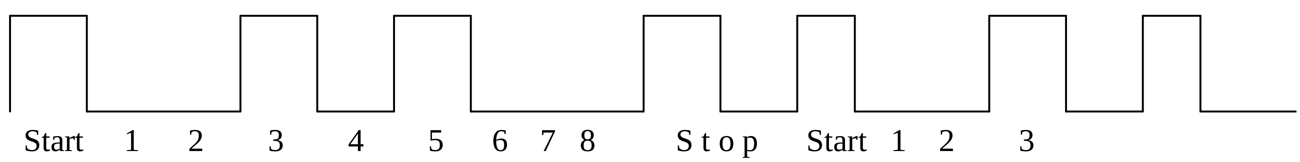
Time (us)	**Input signal 1**	**Output signal 1**	**Input signal 2**	**Output signal 2**
0	**0**	**0**	**0**	**0**
250	**1.414**	**2.828**	**1.9**	**1.9**
500	**2**	**4**	**3.5**	**3.5**
750	**1.414**	**2.828**	**4.6**	**4.6**
1000	**0**	**0**	**5**	**5**
1250	**-1.414**	**-2.828**	**4.6**	**4.6**
1500	**-2**	**-4**	**3.5**	**3.5**
1750	**-1.414**	**-2.828**	**1.9**	**1.9**
2000	**0**	**0**	**0**	**0**
2250	**1.414**	**2.828**	**-1.9**	**-1.9**
2500	**2**	**4**	**-3.5**	**-3.5**
2750	**1.414**	**2.828**	**-4.6**	**-4.6**
3000	**0**	**0**	**-5**	**-5**
3250	**-1.414**	**-2.828**	**-4.6**	**-4.6**
3500	**-2**	**-4**	**-3.5**	**-3.5**
3750	**-1.414**	**-2.828**	**-1.9**	**-1.9**

6. Suppose you are transmitting a light wave down a multimode fiber. Assume that each 1 is encoded as a pulse of light that splits into two signals, one which goes directly down the center of the fiber, the other that bounces off the walls of the fiber, thereby taking a longer route. Suppose the short route takes 100 nanoseconds and the long route takes 125 nanoseconds. What is the maximum bit rate we can use on the fiber without causing intersymbol interference? What happens if we triple the length of the fiber link?

Suppose, we send the first bit at time 0ns. The two signals reach the destination at times 100ns and 125ns. We want that the signals for the second bit should reach after 125ns. Therefore, we can start transmitting the second bit only at time 125-100= 25 ns. Therefore, we can transmit at most 1 bit per 25 ns.
Bit rate = 1/25ns = 40Mbps

In case the length of the fiber triples, the delays also triples. The bandwidth in that case becomes 1/75ns=13.3Mbps

7. Consider an encoding where the clock is resynchronized at the beginning of every byte, using 1 start bit (e.g., transition high), 8 data bits (including a parity bit), and 2 stop bits (e.g., transition high to low). Suppose it is possible to have vicious errors that cause the receiver to completely lose synchronization with the sender. Is there a sequence of characters such that the receiver stays out of synch indefinitely (i.e., the receiver detects a valid sequence of characters but the sender is sending a completely different sequence of characters)?



Suppose the receiver loses synchronization, so that it treats the fifth bit as the start bit. If the sender keeps repeating the data pattern, then the receiver would continuously interpret the 5th data bit as the start bit and the 3rd and 4th bits as the stop bits.

8. Write a program in any convenient language that takes a bit stream (e.g., encoded as ASCII 1's and 0's), and computes an 8-bit CRC on the bit stream, using the polynomial x^8 + x^6 + x + 1 (the bit pattern 101000011).

We will assume that the bit pattern whose CRC has to be calculated is stored in the array *input* and the length of the bit pattern is *len.* Further, the given polynomial is stored in the array *poly.* The following is the C pseudo code:

```
/* Multiply the input by x^8 */
for (i=0;i<8;i++) input[len+i] = 0;

/* Now divide the input by the given polynomial */
currIndex=0;
while (currIndex < len) {
	if (input[currIndex] == 0) {
		currIndex++;
		continue;
	}
	for (i=0;i<8;i++)
		input[currIndex+i] = xor(input[currIndex+i], poly[i]);
	currIndex++;
}
```

The CRC is given by input[currIndex..currIndex+8]

9. Hugh Hopeful is the President of JTBD (Just to Be Different) Network Company. After consulting marketing, Hugh decides to offer a new framing protocol where the user can specify their own flags to delineate the start and end of frames. One popular framing protocol is HDLC, which uses 01111110 as the start/stop signal; to prevent confusion, HDLC stuffs an extra 0 after ever 5 bits in the user data that are all 1's.

Suppose the user provides an 8 bit flag of the form "aSbc" where a, b, and c are arbitrary bits, and S is an arbitrary 5 bit string. Hugh invents the following bit stuffing rule -- whenever you find the string S in the user data, an extra bit equal to the complement of b is stuffed after it. Notice that Hugh's rule gives exactly the HDLC bit stuffing rule. Can Alyssa P. Hacker find a flag that Hugh's stuffing rule will fail on? Explain.

Suppose the 8 bit flag is 11111101, where
a = 1
S = 11111
b = 0
c = 1
Hugh's rule stuffs a 1 after every 5 consecutive 1s. Suppose we send data consisting of a single bit (1) followed by the flag. The receiver therefore receives 111111101. After receiving 5 consecutive 1's, it assumes that the 6th one is bit stuffed. Therefore, it removes it and loses track of the HDLC flag.

10. Consider multiple computers attached to an Ethernet hub (a hub is effectively a multiway repeater between a number of Ethernet segments). The maximum allowable distance between a computer and the hub is 100m. Assume the propagation speed of electrical signals on the Ethernet is 1.75 x 10^8 meters/second. Assume also that the hub takes 100 nanoseconds to detect a collision. What is the maximum time between when a node starts transmitting and when it learns that the packet has collided?

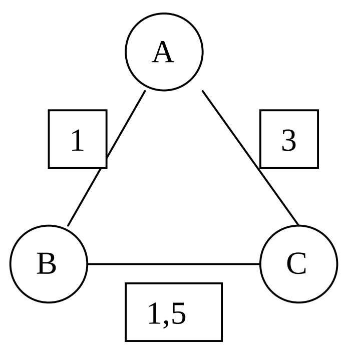
The maximum time between a packet transmission and collision detection occurs when the packet travels to the hub, a collision is detected and then the information about the collision travels back to the sender.
Therefore the maximum time = 2*100/1.75x10^8 + 100 ns
= 1242 ns

11. For each of (i) link state, (ii) traditional distance vector, and (iii) path-based distance vector:

(a) Under what circumstances, if any, is it possible for router A to forward packets for router B through router C, even though routers A and B are neighbors.

For each of the routing protocols, it is possible for A to forward packets for B through C, even though A and B are directly connected, if the cost of the direct link between A and B is higher than cost of the path through C.

(b) Suppose that A and B are neighbors, but A thinks the cost of the link to B is x, and B thinks the cost to the link to A is y. Will this cause any problems? Explain in terms of an example.



In link state protocol, if A and B advertise different cost for the link A-B, eventually everyone would have a consistent view of the topology. Therefore, the inconsistent costs do not cause any problems.

In case of distance vector routing (and path vector), it could lead to unsymmetric routes. If B thinks that the cost of the link B->C is 1, then it will take the direct route to C. However, C would take the path through A to B.

(c) Give a topology in which, due to an inconsistency in the information about a single link, many routes are disrupted. Give a topology in which, despite an inconsistency in the information about a single link, no routes are disrupted.

A large number of links would be affected if the inconsistency is for a link that connects two large networks (figure 1). On the other hand, no routes would be disrupted if the link is at the edge of the network, connecting a leaf node (figure 2).

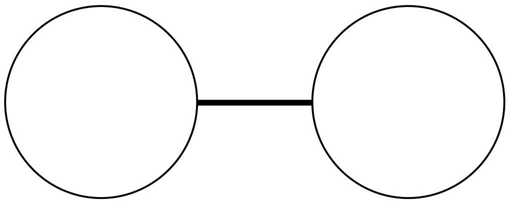


Figure 1

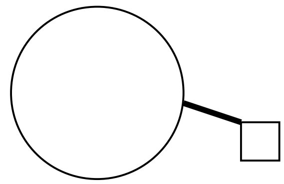
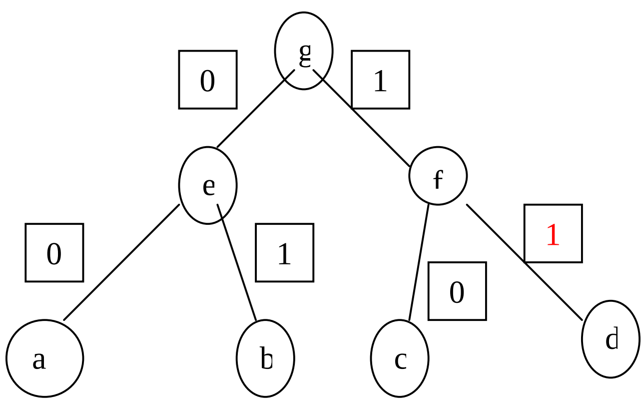


Figure 2

12. Consider an arbitrarily large network in the form of a fully balanced binary tree of L levels. There are 2^k nodes at level k (for k = 0 ... L - 1) and there are 2^L - 1 nodes altogether. Each leaf at the bottom of the tree represents a host and each internal node represents a router. Your task is to devise a scheme for assigning addresses to nodes that is optimal in the sense of minimizing the maximum size of the routing tables across all routers, assuming each entry in the routing tables store only prefixes with unique routes (in other words, the routers are CIDR-based). Indicate what it stored in each router's forwarding table.



We assign L bit addresses to the routers. A node at level k is assigned an address in the following manner:

- **The first k(most significant) bits of the address contain the path from the root node. For instance, the first 2 bits of nodes a,b,c,d are 00,01,10,11 respectively.**
- **The last (least significant) L-k bits follow the pattern 10..0 (a 1 followed by all 0s). Therefore, for levels 0,1,2; these bits are 100, 10 and 1 respectively.**

Using the above rules, the addresses for the various routers are:
a: 001;b: 011;c: 101;d: 111;e: 010;f: 110;g: 100

It is obvious that no two nodes have the same address.
With this scheme, every router has at most 3 entries in the routing table, one for hosts in its left tree, one for hosts in its right tree and the third for all other hosts. Thus, for instance, e would have the entries:

- **00* Next hop: a**
- **01* Next hop: b**
- *** Next hop: g**

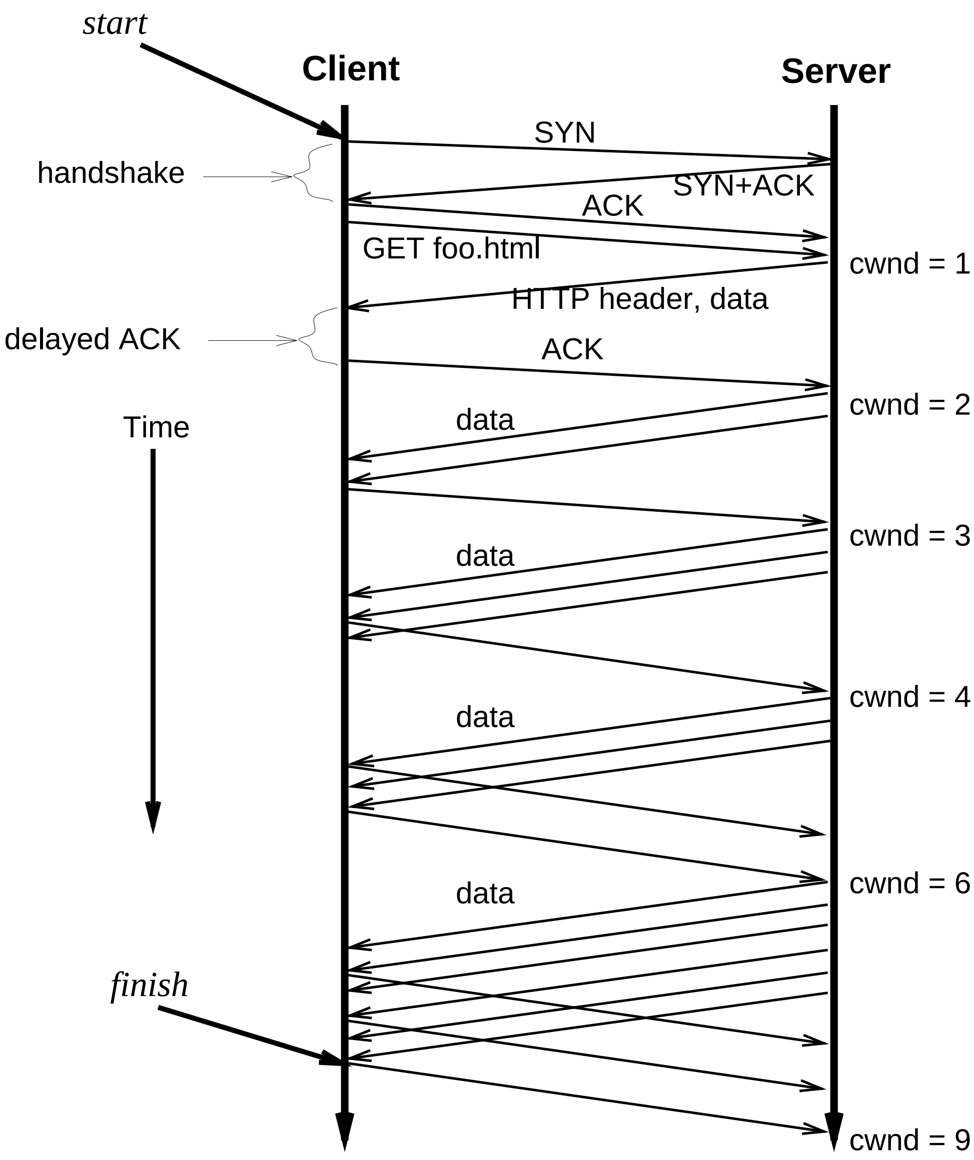
Note that the forwarding algorithm finds the longest matching prefix. So, it uses the * entry only if the destination address does not match the first two entries.

13. Consider the following proposal for hierarchical source routing based on the Internet's "loose source routing". With loose source routing, a host specifies a list of routers to visit on the way to the destination; the network is free to use any means of getting between the routers in the loose source route. A hierarchical source routing scheme could be built on top of this, by saying that when a packet is delivered to a router specified in the loose source route list, it can in turn "push on the stack" a loose source route to the next router in the original list. Explain why this might be better or worse than the way hierarchical routing is done in the Internet today. Where appropriate, give concrete examples.

In comparison to the current hierarchical routing, hierarchical loose source routing is better because it shifts all the decisions to the network edges allowing the core of the network to be dumb. Currently all the routers in the network have to know about the topology and take forwarding decisions. With hierarchical loose source routing(HLSR), only the edge nodes have to know the topology; the internal nodes just have to forward packets based on the specified route. On the other hand, HLSR might have sclability problems with the edges becoming bottlenecks. Further, what happens if one of the routers fails. Failure becomes much less transparent as the complete route is already specified in the packets.

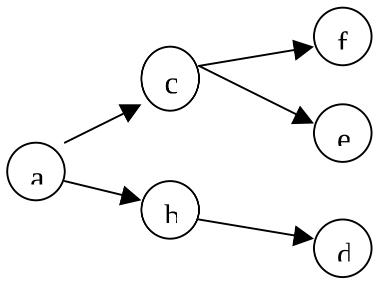
14. Draw a timing diagram illustrating all the packets that are sent between a client and server, when the client requests a 10KB web page, and the previously determined maximum transfer unit without fragmentation is 512 bytes. Include connection setup and teardown, slow start, delayed acks, etc. . Roughly how many round trips does it take? What happens if the third data packet from the server is dropped? How many round trips in this case?

The following figure shows the timing diagram.
It takes 8 round trip times for the complete transfer, as around 20 data packets have to be transmitted.
If the third packet is dropped, the server times out. The duration of the time out depends on the estimate of the round trip time. After timing out, the server reduces its window size to 1 and starts the slow start phase all over again.
Assuming the timeout ~ 2RTT, number of round trips = 9.



15. Recall that in source routing, the sending host lists each hop of the path explicitly in the packet header. Suppose we want to extend unicast source routing to multicast. Sketch a design of a scheme for doing so. How could you efficiently encode the route of a multicast packet in the packet's header? Sketch the format of the packet header that you would define to implement your scheme. Do you think it would be easy to engineer and deploy such an approach in a real network?

For multicast, the whole multicast tree has to be encoded in the message. The message contains the next hops for each of the routers that the multicast packet reaches. Consider the following topology:



The multicast packet would contain the information that b and c are the next hops for a; f and e are the next hops for c and d for b. The message format would be as follows.

	NH	**Host1**	**NNH**	**NH1**	**… NHn**	**Host2**	**… Hostk ….**

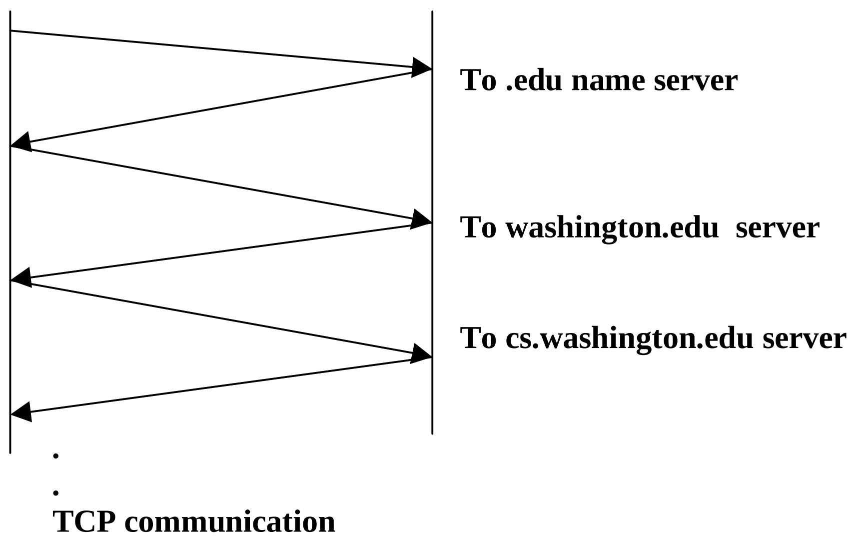
Here, NH is the total number of hosts. NNH is the number of next hops for each of the hosts.
Such a approach would be very difficult to deploy in the network. With bandwidths increasing, the processing time for a packet at the routers becomes crucial. In this case, the processing time could be high as the packet header has to be parsed to extract the relevant information.

16. One difficulty in the deployment of the Mbone has been incompatibilities between variants of multicast routing protocols. Suppose we want to bridge together two multicast networks, N1 running protocol P1 and N2 running protocol P2. Assume there is a sender S in N1 and a receiver R in N2 and that the two networks are interconnected via two gateways G1 and G2. In other words, there are two possible paths between N1 and N2. Suppose that P2 computes a reverse-path route from R to S via G1 but P1 computes the distribution tree from S to N2 via G2. As a result, packets will not be correctly routed from S to R. Explain how this happens. What might you do to avoid this problem?

The problem arises because the multicast packets are routed along a different path (through G2) while the prune messages are propagated along a different path (through G1). Therefore, even though, the receiver R might want to prune the multicast messages, they would be still be sent to it.

A possible solution is to have a rendezvous point at the border between the two networks. The source S sends multicast messages to this rendezvous point and the receiver R also sends prune messages to it.

17. Assuming nothing is cached, use a timing diagram to illustrate the packets that are sent to load my home page, http://www.cs.washington.edu/homes/tom/index.html? How many round trips does it take? What changes if the name is cached but not the page? What if the page is cached? Assume that the request goes through a name proxy and that "edu", "washington" and "cs" are each in their own zone.



The client first goes to the local name server to resolve the DNS name. Since it is not cached locally, the name server returns the address of the .edu name server which in turn returns the address of the washington.edu name server and so on. Note that all these are UDP messages. Once, the client obtains the IP address of the server, it can establish a TCP connection. Packets flow in the same manner as in problem 14. Assuming that the home page is of size 10KB, the number of round trips = 11. Note that here we are ignoring the communication with the local name proxy since it might be on the LAN.

If the name is cached but not the page, it takes 8 RTTs to get the page (prob 14).
In case the page is also cached, then the time is just the time to access the local disk.

18. Recall that DNS server replicas are updated asynchronously, so that at any point in time, some replicas may have seen an update while others have not. Eventually, each update arrives at each replica.

(a) Explain how a node can modify a DNS record, then try to read back the modified record, and still find the old information. This can occur even if no other node has ever modified the record. Why?

This could happen if the node sends the update and read request to different DNS servers. Suppose it sends the update A and the read request to replica B. Then, if the update has not been propagated to B, it would return the stale information.

(b) Sketch a solution that enforces "processor order" -- a node always sees its changes to the database in the order that the node made them (e.g., it never sees changes disappear).

This can be enforced by ensuring that a node always goes to the same replica.

(c) Sketch a solution that enforces a "total order" -- every node sees changes to the database in the same order (e.g., if host A modifies record P then modifies Q, if host B reads Q and gets the new value, if it later reads P, it is guaranteed to get P's new value).

For this we require a global server that assigns "global" sequence numbers to updates. Thus all updates are first sent to the global server. It assigns a unique global sequence number to the update and propagates it. This ensures that all the nodes receive the update in the same order.

19. End-to-end Argument
Almost all network protocols are designed to be robust to failures. For each of the following, explain (i) what would happen if the information was lost, and (ii) how the system would recover, if at all. For example, if one side loses TCP connection state (for example, because it crashes), the TCP connection would be terminated, but the hosts could reestablish communication by making a new connection.

(a) A TCP acknowledgment: **The sender would get an acknowledgment for the next packet. Since TCP uses cumulative ACKs, the new ACK implicitly acknowledges the previous packet. Alternately, the sender could get a timeout, in which case it would retransmit the packet, and get an ACK back.**

(b) A fragment of an IP packet: **The IP packet would be dropped. If the sender and receiver are running a reliable protocol (like TCP) on top of IP, the sender would timeout and retransmit the IP packet.**

(c) A distance vector routing table update: **There could be a temporary routing loop, or packets could be dropped. But the distance vector update would be flooded periodically. Eventually, it would be received by everyone, restoring the correct routing.**

(d) A packet carrying link state routing information: **Again, there could be temporary loops or packet could be dropped. But the packet would be retransmitted, so that everyone eventually receives it.**

(e) An ATM virtual circuit entry: The ATM connection breaks and can be re-established.

(f) A DNS resource record for an mit.edu host cached on a Washington name server**: The Washington name server would contact the mit.edu name server (if its record is cached) and get the record for the host from it. This record is then cached. If even the mit.edu is not cached, then it would go to the .edu name server.**

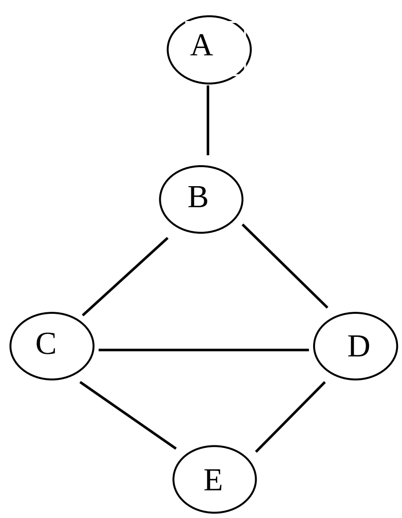
(g) A DNS resource record for an mit.edu host maintained on its home server at MIT.
(Hint: all DNS servers are replicated.)
A host or another DNS server trying to contact the mit.edu server would not get any reply. It would try for a fixed number of times, before trying another replica of the server.

(h) A name/address entry in a host's ARP cache: **When the host wants to resolve a name, it would send a broadcast message. The host with that name would respond with its address, which is then cached in the sending hosts ARP cache.**

(i) Prune-state in a multicast routing tree: **The routers would send the multicast packets to the host, which would resend the prune message. This prune message stops the transmissions and updates the prune state.**

(j) A user's private key in a public key encryption system: **The user cannot decrypt messages that have already been encrypted using its public key by other senders. Eventually, the user gets a new private key and propagates the corresponding public key.**

20. Routing. In this problem, we consider how distance vector update messages interact with multicast. Consider the following topology:



(a) Assuming that each node initially starts with itself in its routing table (with hop count 0), and at each step a single message is sent, what is the state of the routing tables at each node after the following sequence of seven updates is completed: E->D, D->C, C->B, B->A, A->B, B->D, D->E.

The following table shows the state of the routing tables at each of the routers. Each column represents one of the routers. Each cell shows the cost and the next hop for the corresponding destination .

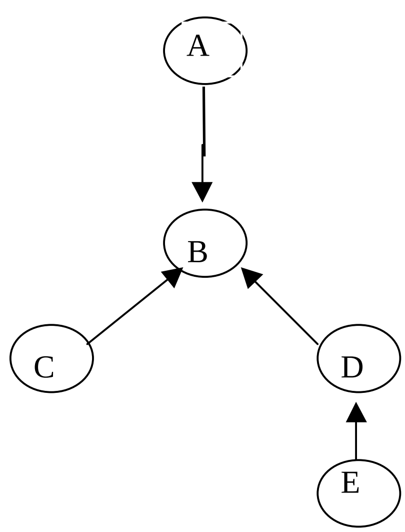
	A	**B**	**C**	**D**	**E**
A	**0,A**	**1,A**	**-**	**2,B**	**3,D**
B	**1,B**	**0,B**	**-**	**1,B**	**2,D**
C	**2,B**	**1,C**	**0,C**	**2,B**	**3,D**
D	**3,B**	**2,C**	**1,D**	**0,D**	**1,D**
E	**4,B**	**3,C**	**2,D**	**1,E**	**0,E**

(b) Show the sequence of packets that that will be sent for a multicast packet originating at node C, using reverse path flooding for the network and routing tables from (a). What happens if shortly after C sends out a multicast packet (but before the multicast has reached all nodes), C sends a routing table update to E?

The following messages are sent:
C->B; C->E; C->D; B->A;B->D; D->E; D->C; E->C

In case C sends a routing table update to E; then the following messages are sent:
C->B; C->E; C->D; B->A;B->D; E->D; D->C; D->E;

(c) Suppose we send a few more distance vector messages, after the C->E message in part (b): D->B, C->D, B->C, E->C. Show the reverse path broadcast tree for multicasts originating at node B.



21. Security. Which is more powerful, public key or secret ("private") key encryption? One way to answer this question is to ask, can we simulate one with the other? Assume we have three parties, A, B, and C, and either each pair shares a secret key (K[A,B], K[B,C], or K[A,C]), or each node has a public key pair, K[A-priv], K[A-pub], K[B-priv], K[B-pub], K[C-priv], K[C-pub], where the private key is known only to the owner, and the public key is known to everyone.

(a) Can we simulate secret key encryption with public key encryption? If so, show how we would encrypt a message from A to B using public key encryption that provides the same security as the message encrypted in K[A,B] -- in other words, the message could only come from A and be read by B. If this is not possible, explain why not.

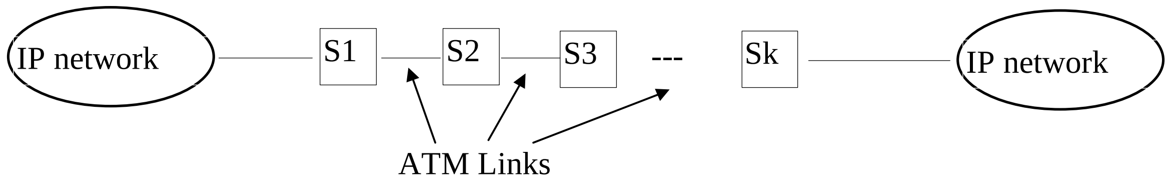
This can be done by encrypting the message with K[A-priv] and then with K[B-pub]. Since it is encrypted with the public key of B, none other than B can read it. Moreover, since it is encrypted with the private key of A, it could have been sent by only A.

(b) Can we simulate public key encryption with private key encryption? If so, show how we can use secret key encryption to provide (i) the same security as a message encrypted only in K[A-priv], and (ii) the same security as a message encrypted only in K[A-pub]. If either of these is not possible, explain why not. You can assume C is trusted by everyone and has a secure disk; for example, it can store messages securely for later retrieval.

(i) **A encrypts the message (m) with K[A,C] and sends it. On receiving the encrypted message, B sends it to C. C decrypts the message to get back m, and encrypts it with K[B,C] along with the information that the message had been send by A. B can now decrypt it and read the message with the assurance that it was sent by A.**

(ii) **Suppose B wants to send a message m securely to A. In public key encryption, it would do this by encrypting m with K[pub]. In shared key, it sends (m,A) encrypted with K[B,C] to C. C returns m encrypted with K[A,C] to B, which then sends it to A. Only A can read this message.**

22. IP switching. Suppose we are using ATM links to carry packets between Internet routers. Here's the scenario. We have **k** routers connected by k-1 ATM links; each link runs at **b** bits per second, the IP packet size is **p** bytes, the ATM cell size is 53 bytes consisting of 5 bytes of header and 48 bytes of useful data, and the propagation delay on each link is **d** seconds. Assume that the average queue length at each router is **q** and that router table lookup takes negligible time. Sample values are k = 4, b = 155 Mb/s, p = 2400 bytes (including the IP header), d = 1 millisecond (0.001 second), and q = 2.



(a) S1 - Sk are IP routers but the links carry ATM cells. When an IP packet arrives that is routed over an ATM link, the packet is fragmented by the router into ATM cells which are then sent over the link. When the cells arrive at the next router, they are reassembled into an IP packet, and the router looks up a route in its IP forwarding table to find the next link. If the packet is routed over another ATM link, it is refragmented back into cells, and transmitted to the next router. Write an expression in terms of **k**, **b**, **p**, **d**, and **q** for the average time for the packet to traverse the k routers, from the time it completely arrives at S1 until it is ready to be sent out the link from Sk into the rest of the Internet. Calculate the result for the sample values above.

NOTE: Assume the queue size to be zero.

Each IP packet is converted to p/48 ATM packets.
Amount of data sent for each IP packet = p*53/48
Time to send each IP packet = T= p*53*8/(48*b)
Time for a packet to reach from Si to Si+1 = T + d
Therefore, total time = (k-1){T+d}
where T = p*424/(48*b)
For the given values, time = 3.41 ms

(b) A number of router vendors have recently added support, called IP switching, to optimize for this case where a number of ATM links are used in sequence. The routers can "bind" an IP flow (sequence of packets) to a virtual circuit that spans multiple routers, in this case, from S1 to Sk. The IP flow is identified by the (IP source address, IP destination address) of the IP header. Once the binding is complete, when a packet arrives at S1, S1 notices that a virtual circuit (such as VC7) has already been set up for that flow, and S1 fragments the packet into cells sent on VC7. When they arrive at S2, the router pretends it is an ATM switch, and forwards the cells as they arrive to S3, and so forth, until they arrive at Sk, where they are reassembled into an IP packet. Assuming the virtual circuit has already been set up, write an expression as in part (a) for the time for a packet to traverse the k routers. Calculate the result for the sample values above.

Since, IP packets are not re-assembled at each router, we can start sending the IP packet as soon as we receive the first ATM cell of the packet. This could also change the average queue length, but for simplicity, we assume that the queue length remains the same.
Time required for each cell to reach from Si to Si+1 = 53*8/b + d
Number of cells = p/48
Time before last cells starts from S1 = (p/48 – 1) 53*8/b
Total time for packet to reach destination = (p/48-1) 53*8/b + (k-1)(53*8/b+d)
= (p/48-1) 424/b + (k-1) (424/b+d)
Plugging in the given values, we get delay = 3.142 ms

(c) Although binding IP flows to ATM connections reduces forwarding delay, the binding process initially incurs a delay at each router to set up the connection. Accordingly, it is not advisable to set up a virtual circuit for every packet that shows up at the edge of an IP. Suppose it takes **T** seconds to set up a virtual circuit at a router/IP switch, write an expression for how big a packet must be for it to make sense to set up a virtual circuit just for that packet. Suppose T = 1 millisecond; would it make sense for the sample values given above? (Of course, the overhead of setting up the virtual circuit can usually be amortized across multiple packets between the same hosts.)

For a virtual circuit to be beneficial

(k-1) (p*424/(48*b) + d) > T + (p/48-1)424/b + (k-1)(d+424/b)

Simplifying, we get

p> $\left(\frac{Tb}{(k-2)424}+1\right)48$

For the given values, p > 8821 bytres

23. TCP. Recently, a number of wireless networks for carrying digital Internet traffic have been deployed; one example is the Metricom packet radio system which can be used for wireless communication on the UW campus. Wireless communication links are typically much more likely to suffer bit errors than traditional "wired" networks, with drop rates approaching 1 in 4 packets. These networks are also usually just for the final "hop"; Internet packets traverse some number of wired links and routers before being sent over the wireless link, which delivers packets directly to the receiving host (for example, a portable PC with a cellular phone and modem).

(a) When these wireless networks were first deployed, observed TCP bandwidth over the wireless link was usually an order of magnitude worse than the peak bandwidth the hardware could deliver. Explain why.

Since the bit error rate of wireless links is very high, TCP experiences a large number of drops on these links. Since TCP congestion control is based on the assumption that drops occur mainly because of queue overflow (which is true for the wired network), TCP mistakes these drops as signs of congestion in the network and backs off by reducing the window size. So, even though there is no congestion, TCP sends at a very low rate which is much below the bandwidth of the network.

(b) One proposed solution to this problem is to cache packets at the upstream router just prior to being sent over a wireless link. If the packet is garbled, the packet can be resent without having to wait for a retransmission from the source host. Of course, this solution is easier to deploy if it doesn't require changing the protocol stack at either the sender or the receiver. Consider the case where the packets are being sent to a wireless host and all the traffic is TCP. Can an intelligent router detect when retransmissions are needed without modifying the receiver? If so, how? If not, why not? Hint: the router can observe TCP acknowledgments.

The last router can buffer the TCP packets and observe the acknowledgments from the receiver. Also, it can have a timeout for every packet that it sent. If the router receives duplicate ACKs or its timer goes off, then it implies that a packet has been corrupted (and not dropped, since this was the last hop). The router can then retransmit the packet. Thus packet losses due to bit corruption are masked from the sender, so that it does not back-off when packets are dropped due to corruption.

More Difficult Questions

24. Derive a formula for the efficiency of a sliding window protocol, with window size X packets, assuming that the packet loss rate (p) is 0, in terms of: T (the timeout value), d (the propagation delay), r (bandwidth of the link), s (packet size), and h (header size).

Round trip time = $2d + \frac{s}{r} + \frac{h}{r}$

Assume that Xs < r*RTT (ie. window size is less than bandwidth-round trip time product)
We are sending X packets in a RTT
Bandwidth achieved = Xs/RTT

Efficiency = Xs/(RTT*r) = $\frac{Xs}{2dr + s + h}$

25. Derive a formula for the efficiency of a go-back N protocol assuming at most one packet is dropped per window (X < 1/p).

Only one packet is dropped from the window. If the first packet is dropped, it implies that all the packets will have to be retransmitted and no progress was made. If the nth packet was dropped then (n-1) packets got through.
Expected number of packets that get through in a RTT
= $p.0 + (1-p)p.1 + (1-p)^2 p.2 + (1-p)^3 p.3 + \ldots + (1-p)^{X-1} p(X-1) + (1-p)^X X$
Simplifying, we get

Average number of packet per RTT = $\frac{(1-p)(1-(1-p)^X)}{p}$

Efficiency = $\frac{(1-p)(1-(1-p)^X)}{2p.d.r}$ **assuming that RTT ~ 2d**

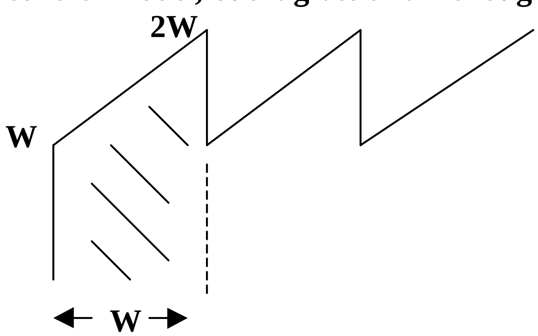
26. Derive a formula for the efficiency of a selective acknowlegment sliding window protocol, assuming the window size is arbitrarily large.

An infinite window size implies that the pipe is always full.
Average number of times a packet is transmitted
= (1-p) + 2p(1-p) + 3p^2 (1-p) + 4 p^3 (1-p) + …..
= 1/(1-p)

Therefore, to send one packet it takes time = s/((1-p)r)
Bandwidth achieved = (1-p)r
Efficiency = (1-p)

27. Derive a formula for the efficiency of a TCP connection, once it has reached steady state, as a function of its packet loss rate (recall that TCP interprets a packet loss as a signal that there is congestion, causing it to reduce its rate by half). Assume at most one packet is dropped in each congestion period (at the tip of the sawtooth), and fast retransmit is used to recover. (Hint: the result should be proportional to the packet size, and inversely proportional to the round trip time and the square root of the packet drop rate). Since the exact solution is complex, it is reasonable to ignore less important terms, but be clear what assumptions you are making.

In the steady state, the TCP congestion window looks like a saw-tooth. The window starts from W/2, increases additively to W; a packet is dropped; and the window drops back to W/2. Note that this is an over-simplification of the TCP congestion control model, but it gives a fair enough idea.



Let the number of packets transmitted before a packet is dropped be m. m is given by the area under the shaded curve.
Thus, m = 3/2 W^2

Probability that m has the value n+1 is the probability that the first n packets are not dropped while the n+1 th packet is dropped = $(1-p)^n$ p
Therefore expected value of m = 1.p+2(1-p)p + 3 $(1-p)^2$ p +4 $(1-p)^3$p + ..
= 1/p
Thus, 3/2 W^2 = 1/p

$$W = \sqrt{\frac{2}{3p}}$$

Bandwidth $= \dfrac{3W^2 s}{2W.RTT}$ **since 3/2W^2 packets are transmitted in W RTTs.**

$= \sqrt{\dfrac{3}{2p}}\dfrac{s}{RTT}$ **where s is the packet size.**

28. One difficulty with using round-trip time estimation to control retransmission timers is accounting for the variability of the actual round trip times. Suppose we use an exponentially weighted moving average to keep track of the estimated round trip time. Let r(n) be the n'th round trip time measurement, and y(n) be the estimated value after n measurements, then we compute: y(n) = a * y(n - 1) + (1 - a) * r(n). Next suppose that we set the timeout to be B* y(n).

If the round trip time is in steady state, with the estimate y(n) equal to the real round trip time z, and the round trip time jumps to a new value s >> z, the system will incorrectly timeout on each packet while the estimate gradually converges to the new value. How long will this convergence take -- how many iterations will it take before the estimate converges close enough to the new value to avoid an unnecessary timeout (B * y(k) > z). Show that when the number of iterations is >= log (1 - 1/B) / log(a), the unnecessary timeouts cease.

Suppose y(0) = s
At time 0, the round trip time changes to z.
Then y(1) = as + (1-a)z
Y(2)= a(as+(1-a)z)+(1-a)z
= a^2 s + z(1-a)(1+a)
Y(3)=a^3s + z(1-a)(1+a+a^2)
In general
Y(n)=a^n + z (1-a)(1+a+a^2+..+a^(n-1))
=a^ns + z(1-a^n)

We want that Y(n)*B>z
B(a^ns + z(1-a^n)) > z
Rearranging, we get

$$n > \log_a\left(\frac{z(1-1/B)}{z-s}\right)$$

Because z-s~z

$$n > \frac{\log(1-1/B)}{\log a}$$

29. In this problem, you need to write some clock recovery code for receiving a 4-5 encoded bit stream. Assume the preamble has been received and the receiver is in sync except for possible clock drift. Thus, the receiver is sampling according to its current clock, but because of clock drift it may be a little off. Remember that in 4-5 coding you are guaranteed to get at least one transition in every 5 consecutive bits; you may get up to 5 transitions, however. (Hint: you may want to receive a group of 5 bits, keep track of how many transitions you sample, and use this to adjust the receiver clock for the next 5 bits. Does it help to oversample -- e.g., take 10 samples to determine if you are missing some transitions?) You can assume there is no noise. Extra credit if you have a sensible strategy to deal with noise.

30. Suppose that each bit in a frame has an independent 50-50 probability of being a 1 or a 0. This might happen, for example, if the data is first encrypted and/or compressed, before being sent. The HDLC bit stuffing rule is to add a 0 after ever 5 consecutive 1's in the user data. What is the average overhead of HDLC? (Hint: The obvious answers like 1/32 or 1/64 are wrong.)

A single 1 is bit stuffed in case the pattern 011111**0 is encountered. (*denotes a done care).**
Two 1's are bit stuffed in case the pattern 01111111111**0 is encountered.**
In general n bits are bit stuffed in case the pattern 01..1**0 is encountered (where the number of 1's is 5n).**
Therefore, the expected overhead
$(1/2)^7\ 1/11 + (1/2)^{12}\ 2/16 + (1/2)^{17}\ 3/21 + \ldots + (1/2)^{(5n+2)}\ n/(5n+6) + \ldots$

31. Unlike unicast addresses, which are semi-permanently assigned to hosts, multicast addresses are often allocated dynamically. That is, a multicast address must somehow choose an address for its traffic -- how does it do this?

(a) One approach would be to create a central authority that would maintain a pool of addresses and "lease them out" when a group of hosts wants to start a multiway communication. Cite two fundamental problems with this centralized approach.

- **The approach is not scalable. The central authority can get overloaded, giving poor performance to the clients.**
- **A central authority also implies a single point of failure. If it fails, then no host can start a multiway communication. Also, the centralized server might become partitioned from a particular host.**

(b) An alternative approach is based on random address allocation. One problem with this scheme is the potential for address collision. Suppose the addresses are allocated randomly from among Z total multicast addresses. If we assume that hosts do not share information about which addresses have already been selected, what is the probability that a collision occurs given N allocations (by N independent hosts)? If Z is 64K, for what value of N does the probability of a collision exceed 1%? Suppose instead that information could be shared on a limited basis. In this case, if we assume that N allocations are carried out each second and each host learns of allocations after one second, and if we assume each address is held for an hour, what is the probability of a collision as a function of N?

Probability of choosing a particular address = $1/Z$
Probability that there are no address collisions = ${}^{Z}C_N\ N!\ (1/Z)^N$

Probability that there is at least on collision = $1 - {}^{Z}C_N\ N!\ (1/Z)^N$

For the second case, a collision occurs if there is a collision during any period of address allocation. The probability of collision during a given interval is same as above.
In the second case, probability of a collision
$= {}^{Z}C_N\ N!\ (1/Z)^N \cdot {}^{Z-N}C_N\ N!\ (1/Z)^N * \ldots\ {}^{Z-3600N}C_N\ N!\ (1/Z)^N$
$= Z!\ (1/Z)^{3600N}\ 1/(Z-3600N)!$
Probability of no collision = $1 - Z!\ (1/Z)^{3600N}\ 1/(Z-3600N)!$