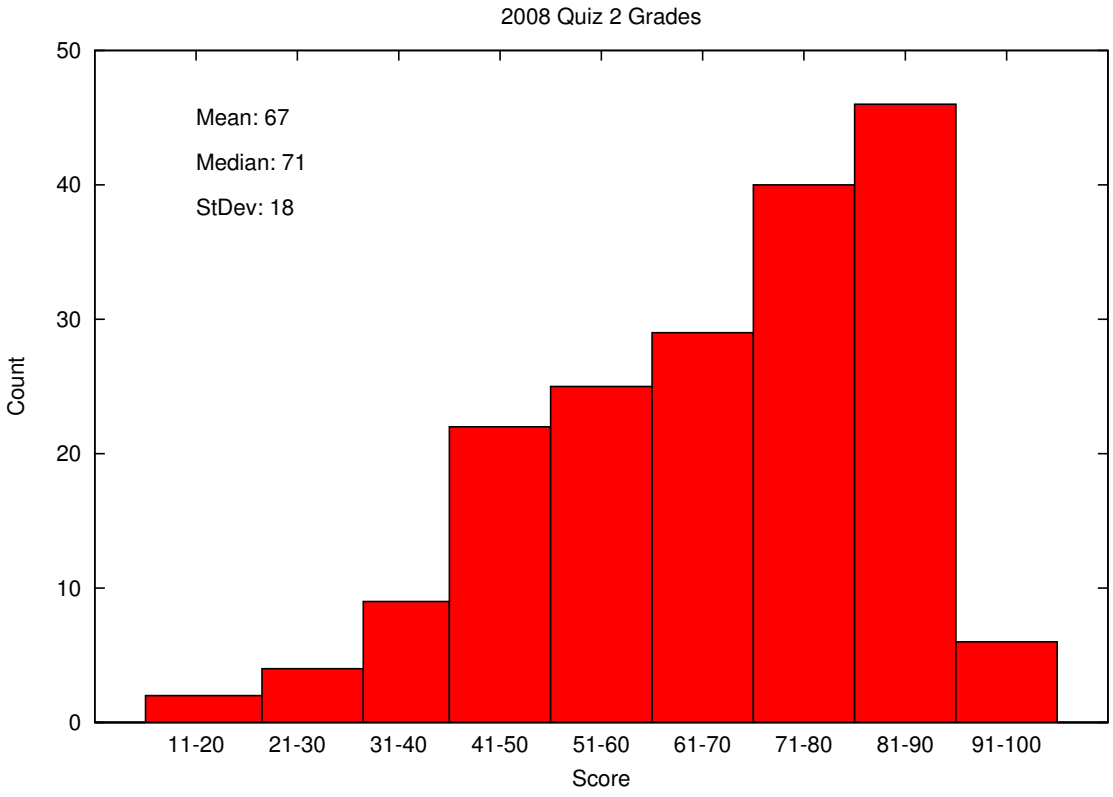
Department of Electrical Engineering and Computer Science

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

6.033 Computer Systems Engineering: Spring 2008

Quiz II - Solutions



I Reading Questions

1. [6 points]: Which of the following statements are true of Ethernet (as described in the Ethernet paper, reading #9)?

(Circle True or False for each choice.)

A. True / False Ethernet enforces a minimum packet size with the intention of minimizing the number of collisions.

FALSE. The purpose of the minimum packet size is to provide reliable collision detection.

B. True / False Using repeaters to overcome signal attenuation, you can make an Ethernet network as long as you want.

FALSE. The maximum size is limited because the RTT must be less than the length of a minimum-size packet. Otherwise the transmitter could complete sending the packet before detecting a collision.

C. True / False Consider one node on an Ethernet network using TCP to send a large amount of data to another node on the same network. If there are no other nodes transmitting on this network, there can't be any collisions.

FALSE. The TCP ACKs can collide with the data packets.

2. [8 points]: Which of the following statements are true of LFS (as described in the LFS paper, reading #15)?

(Circle True or False for each choice.)

A. True / False If a program writes an entire 64KB file in a single `write` system call, all the blocks of the file are likely to be stored in one or two groups of nearby cylinders.

TRUE. LFS will append the data to its log, which it lays out contiguously on the disk.

B. True / False A computer with an LFS file system crashes and then reboots. Data that were written to files after LFS wrote its last checkpoint, but before the crash, are lost; that is, during recovery LFS returns to its state at the time it wrote the checkpoint.

FALSE. During recovery, LFS looks at the end of the log to learn about modifications after the last checkpoint.

For questions C and D, consider a program that uses a single large file that does not change in size. The program performs bursts of reads or bursts writes to random blocks in the file. Assume that during reads, the requested blocks are not found in a RAM cache, and that by the time a burst of writes ends, all the blocks are actually written to disk.

C. True / False If the file is stored on LFS, the disk will perform roughly the same number of seek operations during a burst of random reads as during a burst of random writes.

FALSE. LFS appends each write to the log, which often requires no seeks.

D. True / False If the file is stored on the original UNIX file system, the disk will perform roughly the same number of seek operations during a burst of random reads as during a burst of random writes.

TRUE.

II Geographic Routing

Ben Bitdiddle is very excited about a novel routing protocol that he came up with. Ben argues that since GPS devices are getting very cheap, one can equip each router with GPS capabilities to find its location and route packets based on location information.

Specifically, assume that all nodes in a network are in the same plane and nodes never move. Each node is identified by a tuple `(x,y)`, which refers to its GPS-derived coordinates and no two nodes have the same coordinates. Each node is joined by links to its neighbors, forming a connected network graph. A node informs its neighbors of its coordinates when it joins the network and whenever it recovers after a failure.

When a source sends a packet, it puts the destination coordinates in the packet header. These coordinates play the role of a destination IP address, and you can assume that each sender knows the coordinates of its destination. When a router wants to forward a packet, it checks if any of its neighbors are closer to the destination in Euclidean distance than itself. If none of its neighbors is closer, the router drops the packet. Otherwise the router forwards the packet to its neighbor closest to the destination. Forwarding stops when the packet reaches its destination or is dropped.

3. [8 points]: Circle all that are true about the Ben's geographic routing algorithm.
(Circle ALL that apply)

A. If there are no failures, and no nodes join the network while packets are en route, no packet will experience a routing loop.

TRUE. Each hop brings the packet strictly closer to the destination, or results in the packet being dropped.

B. If nodes fail while packets are en route, a packet may experience a routing loop.

FALSE. A failed node might cause a lost packet, but again each hop either takes the packet closer to the destination or causes it to be dropped.

C. If nodes join the network while packets are en route, a packet may experience a routing loop.

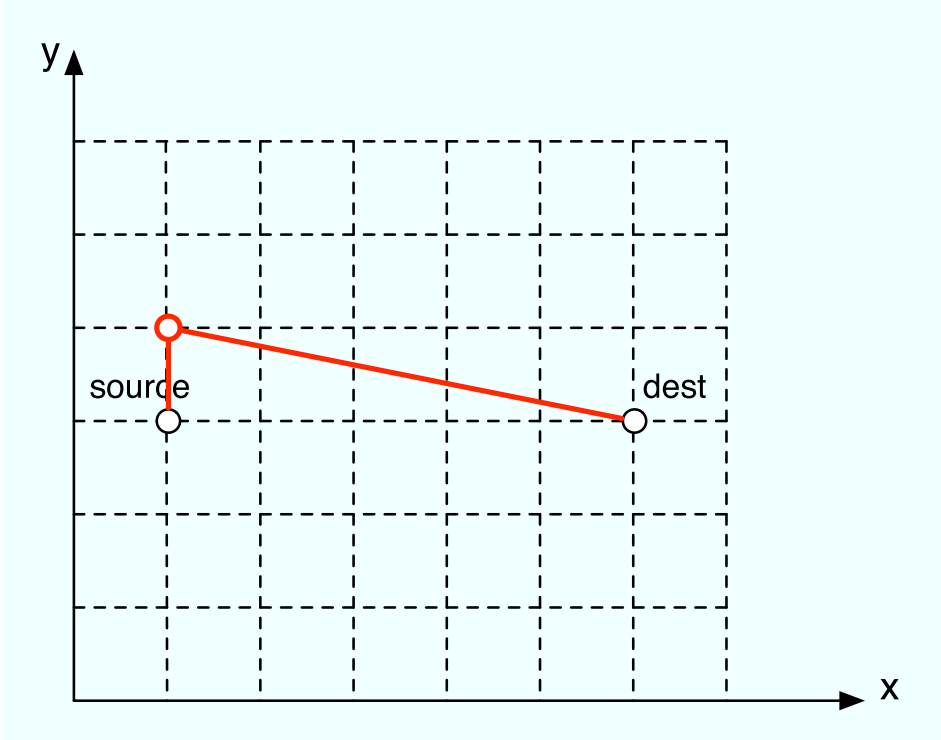
FALSE. Again, a new node will either drop the packet or forward it closer to the destination.

Since Ben's algorithm forces the distance to the destination to strictly decrease, the packet cannot loop.

In the following two questions assume that no link or node fails or joins the network. If drawing an example, do so in the grid provided. Ensure that all nodes are located on the intersection of grid lines, that your graph is connected, and links are clearly marked with solid lines. Links need not follow grid lines.

4. [10 points]: Can Ben's algorithm deliver packets between any source-destination pair in a network? If yes, explain, if no, provide a counter example.

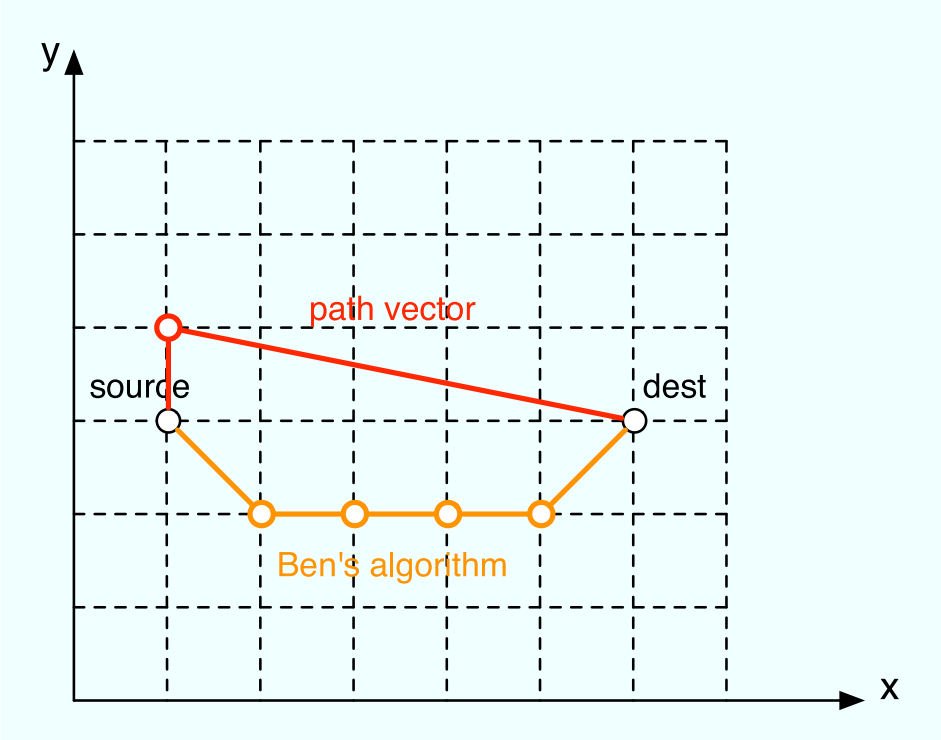
NO. One possible counter example is shown below.



Ben's algorithm will never send packets along the first hop in the path, since this hop travels further away from the destination.

5. [10 points]: For all packets that Ben's algorithm delivers to their corresponding destinations, does Ben's algorithm use the same route as the path vector algorithm discussed in class? If your answer is yes, then explain it. If your answer is no, then draw a counter example.

NO. One possible counter example is shown below.



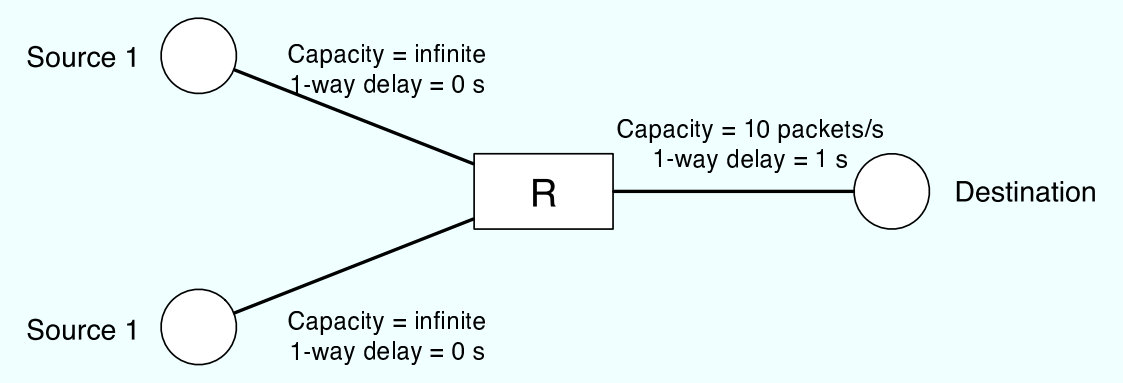
The path vector protocol will select the shorter (in terms of hops) upper path while Ben's algorithm will always choose the lower path.

Acceptable answers included scenarios where both paths were the same length, but where the path vector protocol may choose either path, i.e., where the paths may be different if the two protocols broke ties differently. It was also acceptable to interpret the path vector algorithm as choosing the shortest path based on distance, rather than hop count.

A common mistake was to ignore that the destination is closer to itself than any other node. Hence, if the packet reaches a node that is directly connected to the destination, the node delivers it to the destination and does not drop it.

III Sliding Window

Consider the sliding window algorithm described in the lectures. Assume the topology in the figure below, where all links are duplex (i.e., one wire in each direction) and have the same capacity and delay in both directions. The capacities of the two links on the left are very large and can be assumed infinite, while their propagation delays are negligible and can be assumed zero. Both sources send to the same destination node.



6. [6 points]: Assume the window size is fixed and only Source 1 is active, whereas Source 2 does not send any traffic. What is the smallest sliding window that allows Source 1 to achieve the maximum throughput?

(Please write your answer in the box below)

21

Assuming that ACKs are very small, the time between packet being sent and an ACK being received is the 1-way delay (1 s) + the time to receive a packet at the destination (0.1 s) + another 1-way delay (1 s), i.e., 2.1 s. A window size of 21 packets is required to keep the link fully utilized, given the bandwidth×delay product of 10 packets/s × 2.1 s.

Source 1 does not know the bottleneck capacity and hence cannot compute the smallest window size that allows it to achieve the maximum throughput. Ben has an idea to allow Source 1 to compute the bottleneck capacity. Source 1 transmits two packets back-to-back, i.e., as fast as possible. The destination sends an ack for each data packet immediately.

7. [8 points]: Assume that acks are significantly smaller than data packets, all data packets are the same size, all acks are the same size, and only Source 1 has any traffic to transmit. In this case, Source 1 can compute the bottleneck capacity as follows:

(Circle the BEST answer)

A. Divide the size of a data packet by the interarrival time of two consecutive acks.

TRUE. The receiver sends the first ACK just before it receives the first bit of the second data packet, and sends the second ACK just after it receives the last bit of the second data packet. So the interval between the arrivals of the two ACKs is the data packet size divided by the link capacity.

B. Divide the size of an ack by the interarrival time of two acks.

FALSE.

C. Sum the size of a data packet with an ack packet and divide the sum by the ack interarrival time.

FALSE.

Now assume both Source 1 and Source 2 are active. Router R uses a droptail queue with a very large buffer, i.e., 10 times the number you computed in Question 6.

Source 2 uses standard TCP congestion control to control its window size. Source 1 also uses standard TCP, but hacks its congestion control algorithm to always use a fixed-size window. This window is set to the number you computed in Question 6, which is the smallest size that allows it to maximize its throughput in the absence of other sources.

8. [12 points]: Which of the following is true?

(Circle the BEST answer)

A. Source 1 will have a higher average throughput than Source 2.

FALSE.

B. Source 2 will have a higher average throughput than Source 1.

TRUE. Source 2 will open up its congestion window so that it varies between about 100 and 200 packets. Source 1's window is always 21 packets. Thus between 80% and 90% of the packets in the router's queue will be from source 2, and source 2 will get a corresponding fraction of the link capacity.

C. Both sources get the same average throughput.

FALSE.

IV Logging

The Speedy Taxi company uses a computer to help its dispatcher, Arnie. Customers call Arnie, each asking for a taxi to be sent to a particular address, which Arnie enters into the computer. Arnie can also ask the computer to assign the next waiting address to an idle taxi; the computer indicates the address and taxi number to Arnie, who informs that taxi over his two-way radio.

Arnie's computer stores the set of requested addresses and the current destination address of each taxi (if not idle) in an in-memory database. To ensure that this information is not lost in a power failure, the database logs all updates to an on-disk log. Since the database is kept in volatile memory only, the state must be completely reconstructed after a power failure and restart, as described in Figure 9-21 of the class notes. The database uses write-ahead logging as in Chapter 9-C: it always appends each update to the log on disk, and waits for the disk write to the log to complete before modifying the cell storage in main memory. The database processes only one transaction at a time (there is no concurrency).

The database stores the list of addresses waiting to be assigned to taxis as a single variable; thus any change results in the system logging the entire new list. The database stores each taxi's current destination as a separate variable. A taxi is idle if it has no address assigned to it.

We will only consider one action that uses the database: `DispatchOneTaxi`. Arnie's computer presents a UI to him consisting of a button marked `DispatchOneTaxi`. When Arnie presses the button, and there are no failures, the computer takes one address from the list of addresses waiting to be assigned, assigns it to an idle taxi, and displays the address and taxi to Arnie.

The code for `DispatchOneTaxi` is as follows:

```
DispatchOneTaxi():
  BeginTransaction
    // read and delete the first address in list
    Read list
    if length(list) < 1
      AbortTransaction
    address = list[0]
    delete list[0]
    Write list

    // find first free taxi
    taxi_index = -1
    for i = 0 to NumberOfTaxis
      Read taxis[i]
      if taxis[i] == Null and taxi_index == -1
        taxi_index = i
    if taxi_index == -1
      AbortTransaction
    // record address as the taxi's destination
    taxis[taxi_index] = address
    Write taxis[taxi_index]
  EndTransaction
  print to dispatcher "DISPATCH TAXI " + taxi_index + " TO " + address
```

For all the following questions, you can assume that there is no activity except as described (or necessarily implied) by the question.

`list` contains addresses `a1` and `a2`. There are two taxis (`taxis[0]` and `taxis[1]`) are both are idle (`Null`). Arnie pushes the `DispatchOneTaxi` button, but he sees no `DISPATCH` display, and the computer crashes, restarts, and runs database recovery. Arnie pushes the button a second time, again sees no `DISPATCH` display, and again the computer crashes, restarts, and runs recovery.

9. [10 points]: If you were to look at the very end of the database's log at this point, which of the following might you see, and which are not possible? `Bx` stands for a *begin* record for transaction ID x, `Mx` is a *modify* (i.e. *change*) record for the indicated variable and new value, and `Cx` is a *commit* record.

(Circle ALL that apply)

A. No log records corresponding to Arnie's actions.

YES, if both crashes occur right at the start of DispatchOneTaxi, before it even starts the transaction.

B. `B101, M101 list=a2, M101 taxis[0]=a1, C101, B102, M102 list=, M102 taxis[1]=a2, C102`

YES, if the first crash occurs just after the EndTransaction (before the print), and the second occurs just before EndTransaction.

C. `B101, M101 list=a2, M101 taxis[0]=a1, B102, M102 list=, M102 taxis[1]=a2`

NO. The first transaction didn't commit, so the list cannot be empty after the second transaction.

D. `B101, M101 list=a2, M101 taxis[0]=a1, C101, B102, M102 list=a2, M102 taxis[0]=a1`

NO. The first transaction committed, and must have left the list with one element, so the second transaction must leave the list empty.

E. `B101, M101 list=a2, M101 taxis[0]=a1, B102, M102 list=a2, M102 taxis[0]=a1`

YES, if both crashes occur just before the EndTransaction.

Suppose again the same starting state (the address list contains `a1` and `a2`, both taxis are idle). Arnie pushes the button, the system crashes without displaying a `DISPATCH` message, the system reboots and runs recovery, and Arnie pushes button again. This time the system does display a `DISPATCH` message.

10. [10 points]: Which of the following are possible messages?

(Circle ALL that apply)

A. `DISPATCH TAXI 0 TO a1`

YES, if the crash occurred before the EndTransaction.

B. `DISPATCH TAXI 0 TO a2`

NO. If the first transaction took a1 off the list, it should have allocated a1 to taxi 0.

C. `DISPATCH TAXI 1 TO a1`

NO. The first transaction must have allocated address a1 to taxi 0, hence cannot allocate this address to taxi 1.

D. `DISPATCH TAXI 1 TO a2`

YES, if the crash occurred just after the EndTransaction.

Arnie questions whether it's necessary to make the whole of `DispatchOneTaxi()` a single transaction. He suggests that it would work equally well to split the function into two transactions, one from the `Read` of list to the `Write` of `list` (inclusive), and the other from just before the `for` loop to just after the `Write` of `taxis[taxi_index]`. Arnie makes this change to the code.

Suppose again the same starting state (the address list contains `a1` and `a2`, both taxis are idle). Again, Arnie pushes the button, the system crashes without displaying a `DISPATCH` message, the system reboots and runs recovery, and Arnie pushes button again. This time the system displays a `DISPATCH` message.

11. [12 points]: Which of the following are possible messages?

(Circle ALL that apply)

A. `DISPATCH TAXI 0 TO a1`

YES, if the crash occurred before the first EndTransaction.

B. `DISPATCH TAXI 0 TO a2`

YES, if the crash occurs after the first EndTransaction and before the second BeginTransaction. That is, after taking a1 off the list, but before giving it to taxi 0.

C. `DISPATCH TAXI 1 TO a1`

NO. Apparently the first DispatchOneTaxi allocated an address to taxi 0 and committed its second transaction. In that case it must have committed its first transaction also, leaving only a2 to be allocated to taxi 1.

D. `DISPATCH TAXI 1 TO a2`

YES, if the crash occurred just after the second EndTransaction.