Department of Electrical Engineering and Computer Science

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

6.033 Computer Systems Engineering: Spring 2017

Quiz I

There are **16 questions** and **15 pages** in this quiz booklet. Answer each question according to the instructions given. You have two hours to answer the questions.

- The questions are organized (roughly) by topic. They are not ordered by difficulty nor by the number of points they are worth.
- **If you find a question ambiguous, write down any assumptions you make.** Be neat and legible.
- Some students will be taking a make-up exam at a later date. **Do not** discuss this quiz with anyone who has not already taken it.
- You may use the back of the pages for scratch work, but **do not** write anything on the back that you want graded. We will not look at the backs of the pages.
- Write your name in the space below. Write your initials at the bottom of each page.

This is an open-book, open-notes, open-laptop quiz, but you may **NOT** use your laptop, or any other device, for communication with any other entity (person or machine).

Turn all network devices, including your phone, off.

CIRCLE your recitation section:

10:00	1. Michael/Dustin	7. Karen/Emily	8. Howard/Ben	15. Jing/Paul
11:00	9. Michael/Dustin	10. Karen/Emily	11. Howard/Ben	16. Jing/Paul
12:00	4. Martin/Isabel	12. Mark/Anying		
1:00	2. Martin/Isabel	14. Mark/Anying	5. Melva/Cathie	13. Peter/Xavier
2:00	6. Melva/Cathie	3. Peter/Xavier		

Name: SOLUTIONS

I Virtual Memory and Filesystems

1. [10 points]: Mark's operating system uses a hierarchical page table scheme for virtual addressing. In this scheme, the first m bits ($m \geq 1$) of a virtual address specify the outer page number, the next n bits ($n \geq 1$) specify the inner page number, and the last $p = 32 - n - m$ bits ($p \geq 1$) specify the offset into the page.

m	n	p

A. For this part only, assume that $m = 16, n = 8, p = 8$. An access is made to virtual address `0x00001234`. Knowing only this, fill in as many bytes as possible for the corresponding physical address (you may give the bytes in hex or in binary). If there is not enough information to determine some byte, write ?.

Corresponding physical address: `0x` ? ? ? 34

The offset – 8 bits, 2 hex digits – is the only part of the address that is preserved

In contrast to Mark, Pete's operating system uses a standard page table scheme for virtual addressing. In this scheme, the first $m+n$ bits of a virtual address specify the page number, and the last $p = 32-n-m$ bits specify the offset into the page. Both Pete's and Mark's OSes are using the same values for m, n, and p, but those values are not necessarily the ones used in Part A of this problem.

B. Assuming that there has been at least one access to memory in both OSes, and that all page table entries are k bytes, fill in the following values:

one outer P.T. 2^n inner PTs

Maximum number of bytes allocated for page tables in Mark's OS: $k \cdot (2^m + 2^{m+n})$

Maximum number of bytes allocated for page tables in Pete's OS: $k \cdot (2^m \cdot 2^n) = k \cdot (2^{m+n})$

Note: the same. PTs are allocated contiguously

one outer one inner

Minimum number of bytes allocated for page tables in Mark's OS: $k \cdot (2^m + 2^n)$

Minimum number of bytes allocated for page tables in Pete's OS: $k \cdot (2^{m+n})$

C. Pete wants to increase the number of virtual addresses his processes have access to. Which of the following should he do? Circle all that apply.

(a) Increase the value of $m+n$ and decrease the value of p by the same amount.

(b) Increase the value of p and decrease the value of $m+n$ by the same amount.

(c) Use hierarchical page tables.

[✓ (d)] None of the above.

All access 2^{32} virtual addresses

2. [6 points]: Ben and Melva are performing some basic UNIX operations, using the version of UNIX described in the paper by Thompson and Ritchie.

First, Ben creates a file `file1.txt` in `dir1`. He writes 4000 bytes of data to `file1.txt`.

(a) **True** / [✓ **False**] `file1.txt` is guaranteed to be stored contiguously on disk.

Next, Melva creates `file2.txt` in directory `dir2` (you can assume that `dir1 != dir2`); `file2.txt` is a link to `file1.txt`. Ben deletes `file1.txt` from `dir1`.

refcount ≠ 0

(b) [✓ **True**] / **False** If Melva opens `file2.txt`, it will still contain the contents of `file1.txt`.

Finally, Melva runs the following command: `ls | grep ''data.txt''`.

(c) **True** / [✓ **False**] When Melva ran her command, the process that ran `ls` must have completed before the process that ran `grep` began.

Run in parallel, but grep blocks on some amount of input from ls.

II Threads

For contrast, consider: yes | grep 'data.txt'! If this q. were true, that command would never complete.

3. [4 points]: Zara is working on her code for bounded buffer `send` using condition variables. The version of `wait()` that Zara has access to does not take a lock as an argument to the call; instead, programmers are responsible for releasing/acquiring the appropriate lock on their own. Zara's current pseudocode is below (line numbers are included for convenience).

```
1: send(bb, message):
2:     acquire(bb.lock)
3:     while (bb.in - bb.out == N):
4:         release(bb.lock)
5:         wait(bb.not_full)
6:         acquire(bb.lock)
7:     bb.buf[bb.in % N] = message
8:     bb.in = bb.in + 1
9:     release(bb.lock)
10:    notify(bb.not_empty)
```

Zara knows that the above code is susceptible to the "lost notify" problem. To fix this, she proposes swapping the order of lines 4 and 5. Answer true or false for each of the following.

(a) **True** / [✓ **False**] This swap will introduce a new race condition.

(b) [✓ **True**] / **False** This swap will introduce deadlock.

wait needs bb.lock, which Zara will hold

4. [9 points]: Isabel is building a server to store and manipulate bank account balances. Her server provides several routines:

```
int balances[NUM_ACCOUNTS] // array of accounts

procedure get_balance(account):
    return balances[account]

procedure transfer(account1, account2, amount):
    balances[account1] = balances[account1] - amount
    balances[account2] = balances[account2] + amount
    return amount
```

Clients issue RPCs to the server to invoke `get_balance()` and `transfer()`. Isabel's server—which runs as its own process—handles each request with a separate thread (i.e., one new thread per request).

To demonstrate her server, Isabel writes a graphical user interface (GUI) client that connects to the server and performs `get_balance()` or `transfer()` operations in response to user-supplied commands. Each instance of the GUI runs as its own process, separate from the server.

Isabel's server and GUI are written in a language like C that allows a buggy program to write anywhere in its memory. Isabel's machine has one processor with one core. Isabel runs her GUI and server in separate address spaces on the same machine.

Isabel is concerned that her code might be slow and incorrect, so she comes to you for help. On the next page, Isabel proposes several modifications to her banking application. For each choice, tell Isabel whether it would:

(a) Enforce modularity by making it less likely that bugs in the GUI affect the internal operation of `get_balance()` and `transfer()`.

(b) Improve throughput without introducing additional sources of incorrect results.

(c) Eliminate sources of incorrect results in the presence of multiple simultaneous clients (i.e., GUI instances).

(d) None of the above.

Select only the **best** answer for each proposed modification.

A. Cache the results of RPC calls to `get_balance()` in the GUI, while still running `transfer()` calls on the RPC server. This takes the form of a new client-side RPC stub for `get_balance()`:

```
get_balance_stub(account):
    if (account not in cache)
        cache[account] = result of sending get_balance(acct) to RPC server
    return cache[account]
```

Effect of modification (a, b, c, or d): (d)

Would be (b), but incorrect results (stale caches) are now possible

B. Place a lock around the reads and writes of balances in the server:

```
transfer(account1, account2, amount):
    acquire(balance-lock)
    balances[account1] = balances[account1] - amount
    balances[account2] = balances[account2] + amount
    release(balance-lock)
    return amount

get_balance(account):
    int bal
    acquire(balance-lock)
    bal = balances[account]
    release(balance-lock)
    return bal
```

Effect of modification (a, b, c, or d): (c)

C. Run the account server on a separate machine from the clients (i.e., GUI instances). Assume that RPCs between machines take long enough that this doesn't improve performance.

Effect of modification (a, b, c, or d): (a)

Initials:

5. [6 points]: Anying is executing Eraser on the following piece of code:

```
acquire(lock_a)
acquire(lock_b)
x = x+1
release(lock_b)
y = x+2
release(lock_a)
```

In all below, $C(n)$ refers to the lock set of variable n.

A. Anying executes the simplest version of Eraser on the above code. This version has no improvements for initialization, read-shared data, or read-write locks.

(a) At the end of Eraser's execution on the above code, what are the values of $C(x)$ and $C(y)$?

$C(x)$: {lock_a}

$C(y)$: {lock_a}

(b) **True** / [✓ **False**] Eraser will issue a warning at the end of its execution of this piece of code.

$C(x) \neq \emptyset$, $C(y) \neq \emptyset$

B. Next, Anying moves onto a piece of code that contains the following two functions. It is possible that multiple threads will execute these two functions concurrently.

```
f1():                              f2():
    acquire(lock_a)                    acquire(lock_b)
    acquire(lock_b)                    acquire(lock_a)
    x = x+1                            x = x-1
    release(lock_b)                    release(lock_b)
    release(lock_a)                    release(lock_a)
```

(a) **True** / [✓ **False**] Eraser will issue a warning after Anying executes it on the above code.

This code causes deadlock, but Eraser doesn't check for deadlock.

III Performance

6. [8 points]: Martin's MapReduce system consists of a master machine and four worker machines $M_1, \ldots, M_4$. Each worker machine has four cores, and can run one map or reduce job per core in parallel. Each worker machine M_i also has access to its own local disk D_i.

Martin's MapReduce job starts with sixteen map tasks; the master assigns one task per core.

A. After awhile, all map tasks have completed except `Task 1` running on M_1. What will the master do in response?

Straggler

(a) Nothing, unless M_1 fails.

[✓ (b)] Assign `Task 1` to an available machine.

(c) Subdivide `Task 1` into multiple smaller map tasks, and assign those smaller tasks to available machines.

(d) Kill the process running `Task 1` on M_1 and begin to assign reduce tasks, ignoring the result of `Task 1`

B. Eventually, M_1 fails. This is the only failure in the system. Circle True or False for each of the following.

(a) **True** / [✓ **False**] M_1 will notify the master of its failure.

(b) [✓ **True**] / **False** The master will re-assign any jobs running on M_1 to another available machine.

(c) **True** / [✓ **False**] The master will copy data on D_1 to another disk.

C. Eventually, the master begins assigning reduce tasks. Answer true or false for each of the following.

(a) **True** / [✓ **False**] A reduce task on M_i usually only involves reads from D_i.

(b) [✓ **True**] / **False** A reduce task on M_i only outputs a single file.

(c) [✓ **True**] / **False** In the case of failures, a reduce task may be running at the same time as a map task.

7. [6 points]: DNS clients C_1 and C_2 have a default nameserver N. C_1, C_2, and N have caching enabled, and will cache all DNS responses (including authoritative records and referral records). No other entity in the network has caching enabled, and no entity in the network—including C_1, C_2, and N— has enabled recursive lookups. No other DNS clients communicate with N. There are no failures in the network.

Assume that the round-trip-time between any pair of machines in the network is constant and equal (i.e., all round-trip-times are equal to some value R). The time it takes to do a cache look-up is negligible. All caches are initially empty.

The following sequence of events occurs:

Step 1: C_1 resolves `www.a.com`.
Step 2: C_1 resolves `www.a.com` (again).
Step 3: C_2 resolves `www.a.com`.
Step 4: C_1 resolves `web.a.com`.

time
4R ← C_1 to N; C_1 to root; C_1 to .com; C_1 to a.com.
0 ← cached
4R ← Same as Step 1 b/c N didn't perform the lookup in Step 1
R ← C_1 to a.com.

You can assume that N is **not** the authoritative nameserver for any part of the hostname `www.a.com` or `web.a.com`.

A. Which step will take the longest to complete? Circle all that apply (in the case of ties there may be more than one correct answer).

[✓ (a) Step 1]
(b) Step 2
[✓ (c) Step 3]
(d) Step 4

B. The next day, you wipe the cache on both clients and on N, and enable recursion on N. You then execute the same sequence of steps. Assume that there are no failures in the network. Circle True or False for each of the following.

(a) **True** / [✓ **False**] The time it takes to resolve the request in Step 1 will decrease.
(b) **True** / [✓ **False**] The time it takes to resolve the request in Step 2 will decrease.
(c) [✓ **True**] / **False** The time it takes to resolve the request in Step 3 will decrease.
(d) **True** / [✓ **False**] The time it takes to resolve the request in Step 4 will decrease.

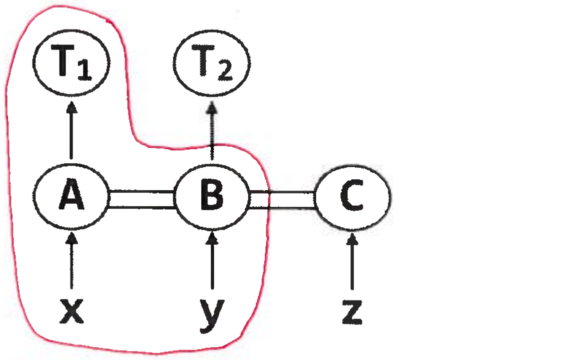
In (a), would decrease in practice b/c of better connections out of N or others using N, but our setup allowed for neither.

In (c) C_2 benefits from recursion + caching @ N

Initials:

IV Routing

8. [~~7~~ 6 points]: Consider the following AS graph. Double-lines indicate peering relationships. Arrows indicate a customer-provider relationship (in $K \rightarrow L$, K is the customer and L is the provider).



x knows about this portion of the network

A. Based on the BGP import and export policies discussed in lecture, answer True or False for each of the following.

(a) [✓ **True**] / **False** x will be able to send traffic to y.

(b) **True** / [✓ **False**] x will be able to send traffic to z.

(c) [✓ **True**] / **False** x will be able to send traffic to machines inside of T_1.

(d) **True** / [✓ **False**] x will be able to send traffic to machines inside of T_2.

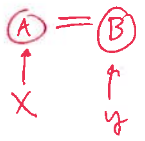
B. Suppose T_1 and T_2 decide to peer. Which of the following is a possible reason to keep the peering agreement between A and B? Circle all that apply.

[✓ (a)] To improve performance when routing between x and y. Shorter path exists w/ A=B

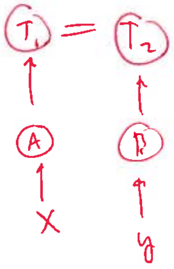
[✓ (b)] For A and B to save money.

(c) To provide connectivity between clients that could not reach each other otherwise (assume there are no failures in the network).

(d) There is no reason to keep the peering agreement between A and B.



makes $ for A & B

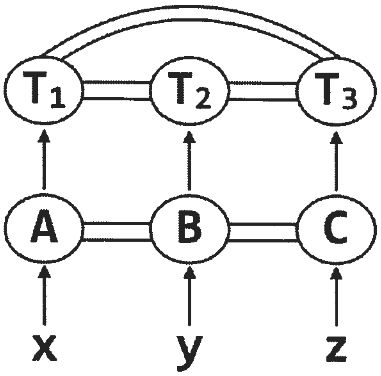


makes them less $ b/c they pay T1 & T2

For (c), connectivity is provided when T1 & T2 peer. A=B provides no additional connectivity.

Initials:

9. [4 points]: Consider the following AS graph.



You are able to place RON nodes in the ASes, and use those nodes to help route data. If a RON node R_1 is placed in AS AS_1, and a RON node R_2 is placed in AS_2, data routed between them will traverse the default BGP path between AS_1 and AS_2. If no such path exists, data cannot be routed from R_1 to R_2.

Jing wants to add RON nodes to the above topology such that data sent from x to z will take the following path: $x \to A \to B \to C \to z$. Which of the following is true? Circle the best answer.

(a) Jing doesn't need to add any RON nodes; data already traverses that path based on standard BGP policies.

(b) Adding two RON nodes—one in A and one in C—will achieve her goal.

[✓ (c)] Adding three RON nodes—one in A, one in B, and one in C—will achieve her goal.

(d) It is impossible to route data via this path.

Default BGP path: $x \to A \to T_1 \to T_3 \to C \to z$

With RONs in A & C, we'd use the BGP path from A to C, which is not $A \to B \to C$ (it's $A \to T_1 \to T_3 \to C$; B won't forward traffic to C on behalf of A)

With RONs in A, B, & C, we take the default path b/t A & B — the peering link — & the same for $B \to C$.

Initials:

V Transport

10. [9 points]: Consider a version of TCP that provides reliable transport, but that does not contain any congestion control mechanisms. Each receiver sends cumulative ACKs to S, as discussed in lecture: an ACK from a receiver R with sequence number x indicates that R has received all packets up to and including x.

A. A TCP sender S sends a window of packets to a TCP receiver R, and receives the following **ACKs** in response.

~~101 102 103~~

```
101, 102, 103, 104, 104, 104, 104, ...
```

101 102 103 104 106 107 108

Assume that S has **not** retransmitted any packets. Based on this sequence of ACKs, which of the following could be true? Circle all that apply. In each case, assume that there were no other losses or delays in the network beyond the one specified.

[✓ (a)] Packet 105 was lost.
[✓ (b)] Packet 105 was delayed.
(c) The ACK with sequence number 105 was lost.
(d) The ACK with sequence number 105 was delayed.
(e) None of the above.

R has not received 105 as evidenced by subsequent ACKs for 104.

B. A TCP sender S sends a window of packets to a TCP receiver R, and receives the following **ACKs** in response.

```
101, 102, 103, 104, 104, 106, 107, ...
```

101 102 103 104 106 105 107

Assume that S has **not** retransmitted any packets. Based on this sequence of ACKs, which of the following could be true? Circle all that apply. In each case, assume that there were no other losses or delays in the network beyond the one specified.

(a) Packet 105 was lost.
[✓ (b)] Packet 105 was delayed.
(c) The ACK with sequence number 105 was lost.
(d) The ACK with sequence number 105 was delayed.
(e) None of the above.

C. A TCP sender S sends a window of packets to a TCP receiver R, and receives the following **ACKs** in response.

```
101, 102, 103, 104, 106, 107, 108, ...
```

101 102 103 104 106 107 108

Assume that S has **not** retransmitted any packets. Based on this sequence of ACKs, which of the following could be true? Circle all that apply. In each case, assume that there were no other losses or delays in the network beyond the one specified.

(a) Packet 105 was lost.
(b) Packet 105 was delayed.
[✓ (c)] The ACK with sequence number 105 was lost.
[✓ (d)] The ACK with sequence number 105 was delayed.
(e) None of the above.

Must have received 105

11. [4 points]: Below is some pseudocode for a portion of a TCP sender. `window` refers to the sender's window, which dictates how many outstanding (un-ACKed) packets it is allowed to have at any given time.

```
for every ACK received
    window = window + 1
```

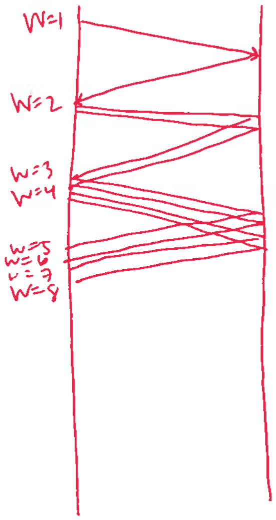
Which phase of congestion control does the above code correspond to?

[✓ (a) Slow start]

(b) Fast-retransmit/fast-recovery

(c) Additive increase

(d) None of the above



Increasing on every ACK gives an exponential increase.

Additive increase would be:

```
In every RTT:
    If no loss:
        window ++
```

The trick is we get > 1 ACK per RTT (in fact, W ACKs per RTT)

12. [4 points]: Consider Deficit Round Robin (DRR) running between two queues, Q_1 and Q_2. Q_1's quantum is x, and Q_2's quantum is $2x$. The maximum packet size in Q_2 is $M > 2x$.

Assume that in a single round of DRR, Q_1 is processed first, then Q_2. What is the maximum number of packets that Q_1 could send before Q_2 sends a single packet?

In R rounds, Q_2 will accumulate $R \cdot 2x$ credit. It's guaranteed to send once $R \cdot 2x > M \Rightarrow R = \lceil M/2x \rceil$

In one round, Q_1 will send at most x packets (with 1 packet per byte). In R rounds, Rx packets

$\therefore \; x \cdot \lceil M/2x \rceil$

13. [6 points]: Three clients, A, B, and C, are sending data into a single queue Q in a round-robin fashion. Q can hold a maximum of ten packets and is using DropTail. Rounds proceed as following:

- If Q is nonempty, a packet is de-queued from the head of Q and sent to a destination D.
- Each client sends a packet into Q. A goes first, then B, then C.

See below for diagram

Cathie wants to make a single change to the above protocol. She is considering the following:

Change 1: Use RED instead of DropTail at Q. The parameters for RED are $p_{max} = 1, q_{min} = 0, q_{max} = 10$.

Change 2: Randomize the order that A, B, and C send in the second step

Change 3: Have the clients use TCP with a fixed window-size of 1 (assume that no congestion-control is in place; each window is set to 1 and never increased). You can assume that ACKs from D return on a separate reverse path; they do not interfere with Q. Moreover, D ACKs any packet it receives immediately, and the ACK is never lost.

A. Which of Cathie's changes would you expect to improve the fairness of the protocol? Circle all that apply.

[✓ (a) Change 1] — Will spread drops across senders

[✓ (b) Change 2]

[✓ (c) Change 3]

(d) None of the above.

B. Which of Cathie's changes will result in the fewest dropped packets? Circle all that apply (there may be more than one correct answer in the case of ties).

(a) Change 1

(b) Change 2 The others have drops; 3 does not

[✓ (c) Change 3]

(d) There is not enough information to decide

Unchanged:

```
                    C B A
                C B A C B
            C B A C B A C
        C B A C B A C B A
C  |  B A C B A C B A C B
CB |  A B A C B A C B A C
CB |  A A B A C B A C B A
```

As time goes on, there is only ever one spot in the queue, and A always gets it.

Initials:

14. [6 points]: Three DCTCP senders, S_1, S_2, and S_3, are in the midst of sending data to various receivers. Currently, all senders have identical window sizes $W > 100$ packets. Each sends a window's worth of data. In the corresponding ACKS, S_1 receives two marks, S_2 receives three, and S_3 receives five.

A. Which sender will decrease its window the most as a result of these marks? Assume that each sender is using the same value of g, and that $0 < g < 1$.

(a) S_1

(b) S_2

(c) S_3

[✓ (d)] There is not enough information to tell

We need to know prior α values

Consider a fourth sender, T_1. T_1 is a TCP sender set to respond to packet marks in the same way that it would packet drops. S_1 and T_1 currently have identical window sizes $W > 100$ packets, and S_1 uses $g = 0.1$. Each sends a window's worth of data. In the corresponding ACKS, S_1 receives two marks and T_1 receives four marks.

B. Which sender will decrease its window the most as a result of these marks?

(a) S_1

[✓ (b)] T_1

(c) There is not enough information to tell

T_1 backs off by ~~at~~ half. S_1 backs off by @ most half regardless of g. It only cuts in half when $\alpha = 1$, which won't happen here.

15. [6 points]: Consider two networks, N_1 and N_2. Both networks have the same topology and all link speeds are identical. The only difference between the two networks is that, in N_1, each switch has a queue that can hold 100 packets; in N_2, each switch's queue can hold 1000 packets.

(a) **True** / [✓ **False**] Assuming the same traffic pattern in both networks, senders in N_2 will always experience higher delay than in N_1.

(b) [✓ **True**] / **False** N_2 can handle larger bursts of traffic from senders than N_1.

(c) [✓ **True**] / **False** TCP senders in N_2 may be able to have larger window sizes than TCP senders in N_1.

Under light load, these networks will perform the same

16. [6 points]: Karen is designing an 802.11 wireless network with two clients—C_1 and C_2—that communicate through an AP, A. All entities in the network can hear each other perfectly.

C_1 and C_2 send data to A. When either C_i successfully sends a packet to A, A will enqueue the packet. A forwards packets from its queue(s) on an outgoing link to a destination D. Both C_1 and C_2 have an infinite amount of data to send.

Karen wants C_1 to send twice as many packets to D as C_2, on average. She is considering two different schemes. In all below, you can assume that a single timeslot is long enough to send exactly one packet. All packets are the same size.

Scheme 1: Use TDMA as the MAC protocol amongst the wireless nodes. Use two different queues at A: one for C_1's traffic, one for C_2's traffic. A will serve the queues in a round-robin fashion, but wait to send two packets from C_1's queue before sending one from C_2's.

Scheme 2: Use "weighted TDMA" as the MAC protocol amongst the wireless nodes: this protocol works exactly as TDMA, except that C_1 gets two slots for every one of C_2's slots. Use a single, unified queue at A. A will serve this queue in a standard first-in first-out fashion.

A. Which of Karen's schemes will enable her goal? Circle all that apply.

[✓ (a) Scheme 1]

[✓ (b) Scheme 2]

(c) None of the above

B. Which of the schemes will cause C_2's packets to experience the most queueing delay at A (the *queueing delay* of a packet at A is the time it spends in A's queue)? Circle all that apply.

[✓ (a) Scheme 1]

(b) Scheme 2

(c) The queueing delay is the same for both schemes

(d) There is not enough information to decide

End of Quiz I

Please double check that you wrote your name on the front of the quiz,
circled your recitation section, and initialed each page.