Spring 2022 Exam 4

You have 70 minutes. The exam tallies up to 100 points. Please be concise: avoid verbose answers. **Good luck!**

Table of Contents:

The Code educates all members of the Georgia Tech Community about the Institute's expectations and Students' rights and creates a standard by which Students are expected to conduct themselves for the purpose of establishing an environment conducive to academic excellence. Georgia Tech Students, Registered Student Organizations, and Groups are responsible for their own behavior, and the Institute has the authority to establish an internal structure for the enforcement of its policies and procedures, the terms of which students have agreed to accept by their enrollment. (Put answers in red)

Question 1 (7 minutes):

We reserve the right to modify numbers for this question on the final exam.

a) We have a disk with the following design specifications:

- Total # of cylinders = **250** (numbered 0-249). Assume that **0** = outermost cylinder and **249** = innermost cylinder.
- Current head position = cylinder **47** and the head is currently moving outwards, towards the disk's cylinder **0**.
- Suppose that the latency of moving the head position is M per cylinder, and the latency of transferring a track is C.
- Assume there are currently **7** pending requests. Assume that each request transfers an entire track, and that after locating the target track, the disk head can immediately start transferring the track (i.e., there is no rotational latency incurred). pg 10-23

Request id:	R1	R2	R3	R4	R5	R6	R7
Cylinder id:	14	17	91	168	56	48	8

Give your answers to questions (i) and (ii) as a function of M and C.

(i) What is the expected response time for R5 (request 5) if the disk scheduling algorithm used is SCAN? Explain your answer. (3 points)

Commented [1]: for question 1, it says the head is moving outwards from 47 to 0, and given some of the requests would appear along the way, are we allowed to have the requests R1 through R7 be handled as their corresponding cylinder ID comes up?

For instance, if the head is moving out, it would first find R2, then R1, then R7. Could we handle them in that order? or do we have to handle them in order of request ID? (first 1, then 2, then 3?) and so on and so forth, until the head reaches 0, then we swap the direction of the head movement?

Commented [2]: we handle them based on their cylinder id

Commented [3]: so we go in order of which id we find first?

Commented [4]: I think so

the head position moves

START -> R2 -> R1 -> R7 -> 0 -> R6 -> R5

47 -> 17 -> 14 -> 8 -> 0 -> 48 -> 56

So if you add all the cylinders it has to cross, you get 103. and then it also read all these tracks (5 tracks) so you get

5C + 103M

Should this be 6C since we also need to re2ad track 0 ?? or we do not have to transfer at track 0 since it is not in the request ?? it should not be 6C because we are not reading track 0. Track 0 is not in the request queue.

(here is work for adding the cylinders it has to cross, correct if wrong):

(47 - 17) + (17 - 14) + (14 - 8) + (8 - 0) + (48 - 0) + (56 - 48) = 103M

abs(47 - 17) + abs(17 - 14) + abs(14 - 8) + abs(8 - 0) + abs(0 - 48) + abs(48 - 56) = 103M

(ii) What is the expected response time for R5 (request 5) if the disk scheduling algorithm used is LOOK? Explain your answer. (3 points)

the head position moves

START -> R2 -> R1 -> R7 -> R6 -> R5

47 -> 17 -> 14 -> 8 -> 48 -> 56

So if you add all the cylinders it has to cross (47-17 + 17-14 + 14-8 + 48-8 + 56-48) you get 87 and then it also read all these tracks (5 tracks) so you get

5C + 87M

b) This question is not related to the previous part (a). Assume the following disk drive specifications:

- Average seek time: 18 ms
- Rotational speed: 7500 RPM
- Platters: 3
- Surfaces: 2
- Tracks per surface: 8000
- Sectors per track: 1024 rot
- Recording density: 256 bytes per sector

This disk has received a request to read 7 random sectors, how much total time (expressed

Commented [5]: why do we start between R2 and R3

Commented [6]: we start at track 47, moving towards track 0 and R2 is just the first one we encounter

Commented [7]: Why is R2 the first one we meet? Is it because it's the next largest value?

Commented [8]: We start from 47 and decrease the numbers (47, 46, 45...so on) so R2, which is 17, comes first in our path

Commented [9]: Why do we go down first and not u

Commented [10]: question says the head is currentl

Commented [11]: My explanation:

Commented [12]: Thanks for the explanation

Commented [13]: in the book there's an example an

Commented [14]: https://www.geeksforgeeks.org/lo

Commented [15]: thanks that makes sense

Commented [16]: How are we getting the 103? if we

Commented [17]: NVM

Commented [18]: I think this work is correct but

Commented [19]: yes. I second this approach

Commented [20]: What's the difference between

Commented [21]: textbook: 10-31

Commented [22]: To put the answer here: it's ironic

Commented [23]: This is similar to a(i) but it doesnt

Commented [24]: Why is it 87 and not 85? I thought

Commented [25]: It would be (47-8)+ (56-8)

Commented [26]: oh ok thanks. And why do we nee

Commented [27]: five tracks we encounter

Commented [28]: How are the tracks counted? Do v

Commented [29]: We move one by one. For the

Commented [30]: Can you give the track numbers th

Commented [31]: *cylinder ids

Commented [32]: R2, then R1, then R7, then to 0,

Commented [33]: We only transfer tracks at the

Commented [34]: What are the 5 tracks for part ii

Commented [35]: the same as part i

Commented [36]: wouldn't it only be 17 ->14->8->48

Commented [37]: so it would be 4C?

Commented [38]: piazza clarification, change ns ->

Commented [39]: Why didn't they update the exam in ms) will that request take to complete? Explain your answer. (5 points)

Avg for 1 sector = average seek time + average rotational latency + time to read 1 sector

= average seek time + 60 / (RPM * 2) + (60 / RPM) / SECTORS_PER_TRACK

18 ms + (60/(7500*2)) * 1000 + (60/(1024*7500))*1000 = 22.0078125 ms

22.0078125 ms * 7 **≈ 154.05 ms**

Commented [40]: Is this because of 2 surfaces?

Commented [41]: no; it'll always be 2 since the rotational latency is half the time it takes the disk to make a complete revolution

Commented [42]: Why are we multiplying by 1000 here? Is that just left over from when it was ns?

Commented [43]: multiply 1000 to change sec to millisec

Question 2 (5 minutes):

Students in CS 2200 are tasked with writing two functions, foo() and bar(), that operate on a shared variable, curr. Both functions are part of a multi-threaded program, thus multiple threads may be calling them concurrently.

The specifications are as follows:

- foo() must set curr to an integer value and call bar(), and
- bar() must increment the value of curr.

a) Student A looks at these specifications and writes the following code:

```
int curr = 0;

int foo(int val) { int y;
      curr = val;
      y = bar();
      return y;
}
```

```
int bar() {
   int x;
curr = curr + 1;
      x = curr; return x;
   }
```

What can go wrong with this code? What is this error called? (2 points)

The problem with this code is that since both foo() and bar() can be called at the same time, our program could be trying to simultaneously set curr equal to val and also read the value of curr in bar(), causing a read-write conflict. This situation where two threads try to access shared data at the same time is also known as a **race condition**.

b) Using the same specifications, student B implements the following code:

```
int curr = 0;
pthread_mutex_t curr_mutex;

int foo(int val) {
    int y;
    pthread_mutex_lock(&curr_mutex); curr =
    val;
    y = bar();
    pthread_mutex_unlock(&curr_mutex); return y;
}
```

```
int bar() {
    int x;
    pthread_mutex_lock(&curr_mutex);
    curr = curr + 1;
    x = curr;
    pthread_mutex_unlock(&curr_mutex); return x;
}
```

What can go wrong with Student B's code? What is this error called? (2 points)

The problem with Student B's code is that when foo() calls bar() in line 7, it has not yet released the curr_mutex variable, and when bar() tries to acquire the lock, it will be stuck waiting behind foo() to release it, causing an error known as a **deadlock**.

C) Suggest how to modify Student B's code to ensure that it produces the desired result (3 points)

In order to modify Student B's code to ensure that it produces the correct result, you could change the location of the call to pthread_mutex_unlock() from after the call to bar() to before the call to bar(), getting rid of the deadlock that would have occurred in the current, unmodified state of the code.

NEW SOLUTION: In order to modify Student B's code to ensure that it produces the correct result we remove the mutex call from bar(). This removes the deadlock problem because bar() no longer waits on foo(). However, with this design choice we must ensure that bar() is always wrapped with a mutex lock and unlock to ensure we do not run into race conditions.

Commented [44]: Or maybe "The situation when a read-write conflict occurs without an intervening synchronization operation separating the conflict is known as a race condition"

Commented [45]: Another foo call could acquire the lock before calling bar() and curr would be changed. Correct solution is to just not lock during bar() >>> (IMO)

Commented [46]: correct me if I'm wrong

Commented [47]: Do you think they just mean in the context of this question? Because moving the location of the call to pthread_mutex_unlock() to before the call to bar() would fix the issue illustrated in the code. But I agree that not locking/unlocking during bar would be another acceptable solution to fixing the deadlock problem.

Commented [48]: The solution currently written will fix the deadlock, but they want "the desired result," which is not guaranteed with the current solution. foo() of another thread could lock the mutex and change curr before bar() of the original thread gets the lock, hence output won't be the expected

Commented [49]: I think that removing the locks from bar would also introduce other issues too since bar can be called independently of foo. A data race condition could then occur if bar is run when foo is overwrite curr.

Commented [50]: but they're also asking for this specific code to run properly, so I kinda agree with Tariq

Commented [51]: I don't think the solution of removing the lock solves the issue of the data race from part a, which is the whole purpose of adapting the mutex lock.

Commented [52]: yeah actually I guess you're right, also the original problem statement says that multiple threads can call foo() and bar() concurrently, so I guess we should change the order.

Commented [53]: Is a conditional lock an option? Pass a different value to bar based on whether it is being called within foo or on its own and apply the lock accordingly

Commented [54]: I disagree with this answer, remember that bar can also be run independently on its own thread. In that case, it would try to write/access curr without a mutex lock, would not give correct result.

Question 3 (8 minutes):

a) Based on Unix commands, determine, for each command, whether a new i-node is created and its reference count. Assume that these commands are executed in the displayed order and that initially none of the files f1, f2, f3, f4 exist in the file system. (6 points)

Command	New i-node? (yes/no)	Old ref count	New ref count
touch f1	yes	0	1
touch f2	yes	0	1
ln -s f1 f3	yes	0	1
cat f3	no	1	1
cp f2 f4	yes	0	1
rm f2	no	1	0

b) What is the difference between ln -s f1 f2 and ln f1 f2? Give one reason you would opt to use the latter instead of the former. (3 points)

~~The "-s" parameter indicates it is a "soft" link and not a hard link, where a hard link produces a new physical copy of the source file f1 under the name of f2. However, a softlink is something that doesn't create a new copy of f1 but rather accesses the same content and space as f1 with a different "tag" or "label" for the user. An example would be like creating a shortcut to an application on your computer. You don't have two copies of it, but two places to access from.~~

The -s parameter indicates a soft link. A soft link differs from a hard link in that the soft link simply references the original file and not the actual data within it. This means that if the original file is deleted, the file that was soft linked is now useless. This is different from a hard link because a hard link points to the same data so even though the original file is deleted, we still can access the data ~~if the original file is deleted, the data persists~~. One reason why we might find a hard link useful is if we want to make sure that our data remains accessible even after the deletion ~~termination~~ of the original file, using a hard link will ensure that the data can still be used.

Commented [55]: generally, is this the total number
Commented [56]: Made some educated guesses
Commented [57]: why would cat increase the ref
Commented [58]: Cat doesn't increase the ref count
Commented [59]: notice that the cat count doesn't
Commented [60]: where do we find this info in the
Commented [61]: ^
Commented [62]: Are these the final answers?
Commented [63]: table at 11-7 in textbook if anyone
Commented [64]: generally, is this the total number
Commented [65]: Made some educated guesses
Commented [66]: why would cat increase the ref
Commented [67]: Cat doesn't increase the ref count
Commented [68]: notice that the cat count doesn't
Commented [69]: where do we find this info in the
Commented [70]: ^
Commented [71]: Are these the final answers?
Commented [72]: table at 11-7 in textbook if anyone
Commented [73]: f3 hasn't been mentioned yet, nor
Commented [74]: For a soft link you create a new i-
Commented [75]: But then what does distinguish it
Commented [76]: I thought a hard link points to the
Commented [77]: no, I am pretty sure Softlink is
Commented [78]: if you see 11-4 page, I think a soft
Commented [79]: weird... I'll update it after reading
Commented [80]: or if someone can correct it while
Commented [81]: generally, is this the total number
Commented [82]: Made some educated guesses
Commented [83]: why would cat increase the ref
Commented [84]: Cat doesn't increase the ref count
Commented [85]: notice that the cat count doesn't
Commented [86]: where do we find this info in the
Commented [87]: ^
Commented [88]: Are these the final answers?
Commented [89]: table at 11-7 in textbook if anyone
Commented [90]: can anyone tell me why it is 0 -> 1
Commented [91]: The table was just updated becau
Commented [92]: generally, is this the total number
Commented [93]: Made some educated guesses
Commented [94]: why would cat increase the ref
Commented [95]: Cat doesn't increase the ref count
Commented [96]: notice that the cat count doesn't
Commented [97]: where do we find this info in the
Commented [98]: ^
Commented [99]: Are these the final answers?
Commented [100]: table at 11-7 in textbook if anyon

(Isn’t the latter a hard link, wouldn't it make more sense to write why a hard link is useful?) A hard link is useful when we want to still be able to access the data in a file even if the original file is deleted. -I agree (Andy Jiang)

Commented [119]: good point!

Commented [120]: and also the fact that access time is shorter right? as suggested by the screenshot below

Therefore, soft links increase usability. On the other hand, every time the file system encounters a soft link it has to resolve the alias by traversing its internal data structures (namely, i-nodes in Unix). We will see shortly how this is done in the Unix file system. The hard link directly points to the internal representation of the original file name. Therefore, there is no time lost in name resolution and can lead to improved file system performance.

https://medium.com/@fermed28/ln-soft-symbolic-links-hard-links-understanding-weird-links-on-unix-604451a0eead

```
HARDLINK
 Old   New
   |     |
 –DATA–

SOFTLINK
NEW -> OLD -> DATA

COPY
Old       New
 |           |
DATA    DATA
```

c) Consider a file system that uses the hybrid storage allocation strategy. The file system has the following parameters:

i) Size of index block = 256 bytes
ii) Size of pointer = 16 bytes
iii) Size of data block = 2048 bytes
iv) Each i-node consists of:
  1) 1 direct data block pointer
  2) 1 single indirect pointer
  3) 1 double indirect pointer

What is the size in bytes of the biggest file that can be created with this file system? (5 points)

Note:The numbers for this problem may change in the actual exam you will be taking.

559104 bytes

# of block pointers (per index block) = 256 / 16 = 16

# of direct data blocks = 1 (1 direct data block pointer)

# of single indirect pointers = (size of index block) / (size of pointer) = 256 / 16 = 16

What if there were 2 single indirect pointers instead of 1 ?

^you'd multiply the data blocks from a single indirect pointer by 2; so in this case, you'd have 32 total data blocks with 16 data blocks from each single indirect pointer.

# of double indirect pointers = 16 * 16 (#singles * #pointers)

What if there were 2 double indirect pointers instead of 1 ??

^same as above for having 2 single indirect pointers; multiply data blocks of 1 double indirect pointer by 2

# of data blocks = 1 + 16 + 16^2 = 273

bytes = 273 blocks * 2048 bytes/block = 559104

What would be the minimum size of the kile??

NEEDS VERIFICATION - NOT SURE IF THE FOLLOWING EQUATIONS CORRECT

I'm not sure if the above equations capture everything. I read the example from the book but am unsure if I understood it entirely. Here are th e equations I came up with from what I interpreted from the book (pg 11-20). If they are correct, I think it would be useful to know in case the numbers are changed:

Max file size in blocks =
number of direct data blocks + number of data blocks with one level of indirection + number of data blocks with two levels of indirection.

Equations for each of these values:

Number of direct data blocks = Number of direct data block pointers *(given)*
Number of data blocks with one level of indirection = (number of single indirect pointers) * (size of index block / size of pointer)
Number of dablocks with two levels of indirection = (number of double indirect pointers) * (size of index block / size of pointer) * (size of index block / size of pointer)

Commented [121]: i dont doubt this is right but why is it 1 direct pts

Commented [122]: How was this calculated? Is it just the number of block pointers?

Commented [123]: I assume so, as per the textbook. see 11-16

Commented [124]: actually, 11-19 would be better

Commented [125]: how do we know there are 16 pointers?

Commented [126]: I believe that it's (size of index block) / (size of pointer) so 256 / 16 = 16.

Commented [127]: ^I agree with this, this is how it is shown in the book.

Commented [128]: 1 indirect pointer points to a index block, since the that index block can hold up to 16 (size of index block/size of pointer = 256/16 =16)direct data block pointers therefore single indirect pointer add another 16 direct data blocks to the size.

Commented [129]: when would these values ever be different?

Commented [130]: afaik, they're always the same bc its a pointer to an index block, whose indices point to other index blocks and the size of an index block is constant

Commented [131]: this could technically be pointers^2

Commented [132]: I made equations to show what I'm thinking based off of the textbook example on page 11-20. if its correct it would be helpful to know if they ...

Commented [133]: What if there were 2 double ...

Commented [134]: I would like to know this too

Commented [135]: then it would be just 2 (16 * 16)

Commented [136]: Can someone answer this ...

Commented [137]: That would just make # of data ...

Commented [138]: I made equations to show what I ...

Commented [139]: awesome thanks so much

Commented [140]: Can someone explain this ...

Commented [141]: That is the calculation for the tota ...

Commented [142]: 0 bytes, touch does this ...

Commented [143]: Can someone verify this

Commented [144]: Almost. ...

Commented [145]: So for one level of indirection/two ...

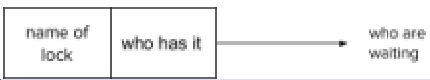
Commented [146]: yea, it would not make sense to (...

Commented [147]: @Aarsh could you explain why?

Commented [148]: Can you explain why you would ...

Question 4 (8 minutes):

The internal representation for a lock in the thread library is shown below:

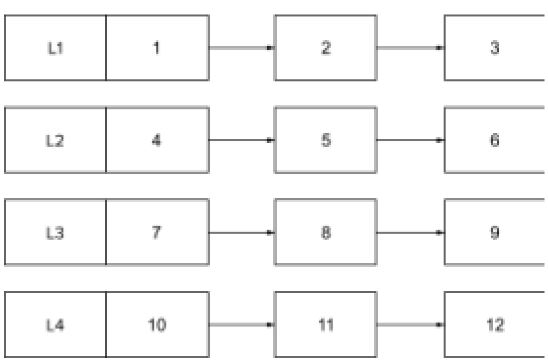


Assume that we have a program with 8 threads, T1 to T8, and the following sequence of events happens:

a) T1 executes thread_mutex_lock(L1);
b) T2 executes thread_mutex_lock(L3);
c) T3 executes thread_mutex_lock(L1);
d) T4 executes thread_mutex_lock(L2);
e) T5 executes thread_mutex_lock(L1);
f) T6 executes thread_mutex_lock(L4);
g) T1 executes thread_mutex_unlock(L1);
h) T7 executes thread_mutex_lock(L2);
i) T2 executes thread_mutex_lock(L4);
j) T8 executes thread_mutex_lock(L3);

Assume that before the start of the sequence, all locks (L1 to L4) were free.

a) Show the thread library's lock bookkeeping state after the aforementioned 10 steps. For each number in the figure (1 to 12), enter the corresponding thread id (T1 to T8). If a number does not have a corresponding thread id, enter N/A. (8 points)
Note: You should put at most ONE thread-id corresponding to each number.



- s

Commented [149]: Can someone explain to me how this sequence of lock calls can happen? L1 is locked three times in T1, T3, and T5 before T1 unlocks L1 at line g. How can multiple Threads share the same lock? Wouldn't this result in a deadlock? like in question 2?

Commented [150]: Great question! I would refer to section 12.2.5 and example 12.9 in addition to example 12.11 in the textbook for more help understanding it. It provides great visuals that make it make more sense.

However, essentially, even though there is one lock and there are three threads all wanting the same one, they wait in a waiting queue until the lock gets released and sequentially passed down the list. The waiting threads do not deadlock but instead are blocked, or basically pause execution until they can received the lock and enter the critical section.

This would not be a deadlock because a deadlock would be a scenario in which several threads are waiting for the same lock but the lock never gets unlocked, therefore keeping any other the threads of the waiting queue from doing any work whatsoever. However, in this case, the L1 does get unlocked.

Theoretically, if you did want to look at these examples with the idea of deadlock versus blocking, L2's waiting queue would be a good example of a deadlock with the assumption that T4 does not release the lock. But again, this is just one section of all of the code, so we ...

Commented [151]: It really does. Thank you!

Commented [152]: T2 is already using L3 so does it just get added to the L4 waiting queue or not since a thread can only be on one waiting queue at a time. Don't know if holding the lock is considered being part of the waiting queue, so thats where my confusion is.

Commented [153]: They are different locks. So i think you can have multiple of them. Consider you have different shared resources and each of them uses a different lock.

Commented [154]: T2 is already using L3 so does it just get added to the L4 waiting queue or not since a thread can only be on one waiting queue at a time. Don't know if holding the lock is considered being part of the waiting queue, so thats where my confusion is.

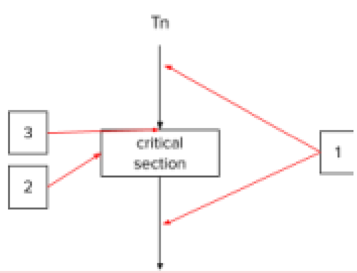
Commented [155]: They are different locks. So i think you can have multiple of them. Consider you have different shared resources and each of them uses a different lock.

Commented [156]: what do the numbers 1-12 represent, bc it sais there are only 8 threads and 4 locks (nothing up to 12)

Commented [157]: So there are 12 numbers in the figure, basically you are filling those numbers(boxes) with the thread IDs, an explanation of the diagram is at the top

L1	T3	→	T5	→	NA
L2	T4	→	T7	→	NA
L3	T2	→	T8	→	NA
L4	T6	→	T2	→	NA

b) Shown in the figure below are possible points of execution for each of the eight threads (T1 to T8). For each thread, select the number that best corresponds to its current point of execution based on your answer to part (a). (4 points)



Note: the execution point numbered "1" corresponds to a thread being before or after the critical section in its execution.

a. T1: 1 // already past critical section finishing execution because unlocked

b. T2: ***see below**

- 2 (because blocked in execution with the idea that critical sections can be nested, so while executing within critical section oaf L2, thread is getting blocked)
- OR 3 (because threads can't have more than one critical section at a time, so T2 is blocked before L3 critical section with assumption it somehow left/unlocked L2 critical section)
- (I think it is 2) - Andy Jiang, because T2 is still in the middle of a critical section even when though it is waiting for a lock

c. T3: 2 // actively executing b/c T1 unlocked L1

d. T4: 2 // actively executing

e. T5: 3 // blocked in execution

f. T6: 2 // actively executing

g. T7: 3 // blocked in execution

h. T8: 3 **** contingent on what's happening with 7-12 in part a**; blocked in execution

Commented [158]: They made a clarification for this question recently.

Commented [159]: Did anyone ask the TAs whether this can be both?

Commented [160]: The critical section could either:
1. depend on both locks being obtained
2. depend on L3 to enter and require L4 partway through

I don't think there's enough info to decide which one it is; I think we need a TA clarfication

Commented [161]: "For each thread select the number (from 1 to 3) that best corresponds to its current point of execution based on the execution sequence above. Show each thread's point of execution with respect to each of the four critical sections, using the following notation: X1(Ly1), X2(Ly2), ...; where Xi is the number between 1 and 3, and Lyi is one of {L1, L2, L3, L4}."

Commented [162]: Yea, so the new answer for T2 would be 2(L3) and 3(L4) I think until L3 is released with assumption that T2 holds the L3 lock

- Question 4b: For the sake of precision:
"For each thread, select the number that best corresponds to its current point of execution based on your answer to part (a)"
is rephrased as follows:
"For each thread select the number (from 1 to 3) that best corresponds to its current point of execution based on the execution sequence above. Show each thread's point of execution with respect to each of the four critical sections, using the following notation: $X_1(Ly_1)$, $X_2(Ly_2)$, ...; where X_i is the number between 1 and 3, and Ly_i is one of $\{L_1, L_2, L_3, L_4\}$."

Does this mean the answer for like T1 would be $1(L_1)$, $1(L_2)$, $1(L_3)$, $1(L_4)$???

Answers based on new format:
T1: 1(L1), 1(L2), 1(L3), 1(L4)
T2: 1(L1), 1(L2), 2(L3), 3(L4)
T3: 2(L1), 1(L2), 1(L3), 1(L4)
T4: 1(L1), 2(L2), 1(L3), 1(L4)
T5: 3(L1), 1(L2), 1(L3), 1(L4)
T6: 1(L1), 1(L2), 1(L3), 2(L4)
T7: 1(L1), 3(L2), 1(L3), 1(L4)
T8: 1(L1), 1(L2), 3(L3), 1(L4)

Commented [163]: Why 1(L2), 1(L3), and 1(L4)? Are those really necessary?

Commented [164]: On piazza it said show the thread's point of execution with respect to each of the four critical sections, so I assumed that meant to show the threads point of execution with each lock, so if it isn't in the queue for the lock its at execution point 1.

Commented [165]: Why is this 3 instead of 2?

Commented [166]: T5 is trying to get lock1 but T3 has it at the moment

	L1	L2	L3	L4
T1	1	1	1	1
T2	1	1	2	3
T3	2	1	1	1
T4	1	2	1	1
T5	3	1	1	1
T6	1	1	1	2
T7	1	3	1	1
T8	1	1	3	1

Question 5 (8 minutes):

The following timelines show the execution history of two threads, each running on a different core of a multicore processor that is endowed with hardware cache coherence for shared memory. x and y are shared memory locations the two threads are operating on, while R1 and R2 are general-purpose registers in each core. For each sequence, determine the final value of x and y.

- Threads:

Thread 1 (T1)	Thread 2 (T2)
Time 0: R1 <- 0 Time 2: R1 <- R1 + 2 Time 4: x <- R2 + R2	 Time 1: R2 <- 0 Time 3: R2 <- R1 + R2 Time 5: y <- x + R1

a. What is the value of x? (2 points)

i. 2

ii. 4 - Correct?

iii. 6

iv. None of the above

b. What is the value of y? (2 points)

i. 2

ii. 4

iii. 6 - Correct?

iv. None of the above

- Threads:

Thread 1 (T1)	Thread 2 (T2)
Time 0: R1 <- 0 Time 1: R1 <- R1 + 2 Time 4: x <- R2 + R2	Time 1: R2 <- 0 Time 2: R2 <- R1 + R2 Time 3: y <- x + R1

a. What is the value of x? (2 points)

i. 2

ii. 4 - Correct?

iii. 6

iv. None of the above

b. What is the value of y? (2 points)

i. 2

ii. 4

iii. 6

iv. None of the above - Correct? -NO (Read piazza)

Commented [167]: Doesn't this mean that as commands are executed, the threads will be using their own local versions of the registers? I ask this because I got a different answer for the first question.

EDIT: Piazza post clarifies that R1 and R2 are NOT registers but memory like x and y.

Commented [168]: Not sure. If they aren't shared, we wouldn't have the ability to see their values and use them in the two threads. So general purpose I assume means they are general purpose use in each core but they are shared.

Commented [169]: Work please?

Commented [170]: Assuming R1 and R2 are shared resource, R1 becomes 0, then R2 becomes 0. Next, R1 is set to 2, then R2 is set to R1 + R2 (2 + 0) since they are not happening simultaneously. Then X is set to 2 * R2 (2 * 2), and then y is set to x + R1 (4 + 2)

Commented [171]: I reordered the lines based on their times so that we know which are happening when and at the same time

Commented [172]: Is this because x is Null?

- Threads:

Thread 1 (T1)	Thread 2 (T2)
Time 0: R1 <- 0 Time 2: R1 <- R1 + 2 Time 3: x <- R2 + R2	Time 1: R2 <- 0 Time 2: R2 <- R1 + R2 Time 3: y <- x + R1

a. What is the value of x? (2 points)

i. 2

zaii. 4

iii. 6

iv. None of the above - Correct? Not just because it could be 0, but strictly because the result of whatever happens at time 2 is nondeterministic <- can someone confirm

b. What is the value of y? (2 points)

i. 2 - Correct? Yes. - how do we have any reason to believe that its 2? We do not know what happens at time 2 (vote for this one)

ii. 4

iii. 6

iv. None of the above - I think this is correct -Andy Jiang - disagree jakob

With the following assumptions:

- Registers R1 and R2 are considered shared memory just like x and ny
- unfilled memory spaces being null and therefore leading to inconclusive answers

I got the following solutions:

a. (x,y) = (4, 6)
b. (x,y) = (4, 2)
c. (x,y) = (0, 2) can we even determine this

These are with Piazza clarifications implemented.

Vote between 2 or none of the above for the value of y for the last part????

2At time 2: reading from the memory: R1 = 0, R2=0 ⇒ R1 = R1+2 = 2, R2 = R1+R2 = 0
At time 3: reading from the memory: R1 = 2, R2 = 0, x = 0 ⇒ x = R2+R2 = 0, y = x+R1 = 2
If you think x = 0 at the end, it y have to be 2 with the same logic
What else can y be?

None of the above <- Guys, I think this is more valid. The point of a question like this is to test our knowledge of race conditions. (-Andy/Fire Lord Zuko) < - No, as stated above in the comments race condition doesn't destroy the output of the program, so if they are testing this knowledge we should see that the output will be 2 not none of the above. Disagree Jakob.

Commented [173]: I think so, yeah.

Commented [174]: There was a clarification on piazza that said shared memory locations were originally zero, so I don't know if these answers are actually correct

Commented [175]: Gotcha. Glad we got clarification. Will rework as soon as possible, but feel free to edit if you have a solution.

Commented [176]: if we assume x and y are originally 0, then, R2 becomes 0 still, R1 becomes 2, and y becomes 2, x remains 0, x is none of the above, and y is 2. Right?

Commented [177]: If we are talking about the second one, at time 2 doesn't R2 become R1 + R2 = 2? So then I agree y would equal 2 but I think x would equal 4

Commented [178]: Assuming x and y are initialized at zero, I agree it would be x=4, y=2

Commented [179]: can someone explain this? why wouldnt x be 4? i feel like y would have an error if ...

Commented [180]: R2 = 0 (assuming at time 2, ...

Commented [181]: I believe that its none of the abov ...

Commented [182]: Yeah since these are memory ...

Commented [183]: I think it is because two threads ...

Commented [184]: I think the question breaks down ...

Commented [185]: I agree

Commented [186]: Is that even possible? Even if the ...

Commented [187]: It's indeterminate. At time 2, thre ...

Commented [188]: why none of the above?

Commented [189]: the way I understand it: at time 3 ...

Commented [190]: I think its because theres a race ...

Commented [191]: But if we were following that, the ...

Commented [192]: I believe that race conditions just ...

Commented [193]: so having a race condition would ...

Commented [194]: yeah, I think 2 is correct as well

Commented [195]: But can't R2 be 0 or 2 at time 2 ...

Commented [196]: Yeah, I agree. I think the value o ...

Commented [197]: the times being the same implies ...

Commented [198]: I believe because the variable is ...

Commented [199]: Dang, I think it may be ...

Commented [200]: Vote

Commented [201]: Same

Commented [202]: At t=2, R1 has to be 2 but R2 ca ...

Commented [203]: I believe this is incorrect. How ca ...

Question 6 (4 minutes):

(a) (3 points) The Ethernet protocol is ubiquitously used for the Media Access and Control (MAC) layer that detects collisions and retransmits a packet if there is a collision. Given this mechanism, why do we also need the transport layer to deal with packet retransmissions? zcw

THIS IS FROM RAMA'S SLIDES (LOOK DOWN, THE ANSWER SHOULD BE OBVIOUS)

The transport layer's main function is to packetize data and messages to be transported along the network. With doing so, we need a protocol on how to handle timeouts, missed packets and packets being received out of order in the sending of packets.

Although the Ethernet protocol provides measures for collision detection and retransmission in the case of a collision, the transport layer is still necessary in order to deal with packet retransmission as a way of addressing any packet loss that may occur while sending the packets.the transport layer is able to do this with actions such as checking source and destination checksums, forward error connection, and more

I think this question is looking for something like collisions are ethernet's job while packet loss is transport's job. There are other things that require resending packets besides collisions.

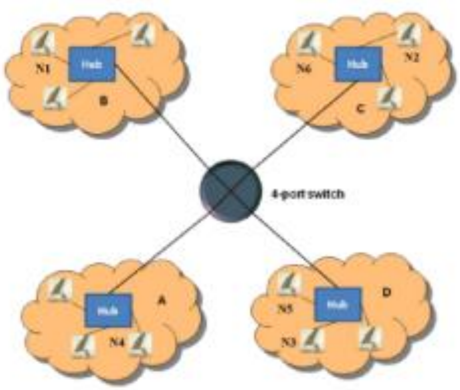
Addition: the transport layer is able to do this with actions such as checking source and destination checksums, forward error connection, and more. (//Just to provide more justification!)

(b) (3 points) In the ethernet network shown in the following figure, assume that nodes N1, N3, N4, and N5 all simultaneously attempt to send packets to N6. Which nodes will experience collision of their packets? Why?

Commented [204]: Not sure if my response is sufficient for this question. I tried to provide the reasons for protocols existing in the transport layer?

Commented [205]: I agree with this

Commented [206]: See 13-63: I think this was what they were going for,
However, in reality, while we say TCP/IP in the same breath thanks to the popularity of the Internet, TCP does not have to run on top of IP. It could use some other network protocol such as ATM.



Since there is a 4-port switch at the center of this setup, N1 and N4, and will not experience a collision, as it has buffers to handle which go first to N6. However, since N3 and N5 are in the same collision domain, they will both experience a collision and thus have to be handled beforehand.

Correct me if I am wrong?

The hosts (N1 to N6) never experience a collision unless they happen to be in the same collision domain (textbook 13-60). something out? N3, N1, and N4 are all sending packets to N6, right? That means something has to be done, as the ethernet can only handle one packet at a time, right?

The switch will handle such collisions: The bridge recognizes the conflict and serializes the traffic flow between the two sides. For this reason, a bridge contains sufficient buffering to hold the packets when such conflicts arise (textbook 13-60).

Sid: Does that mean only N3 and N5 collide? I think so.

Sid: Then what happens when all 4 hubs send? That doesn't happen in this case

Question 7 (12 minutes):

a) (8 points) In the network below, each link is bi-directional and symmetric. With the indicated link costs, use the Distance Vector algorithm to compute the shortest path from network node y to all other network nodes. Show the contents of the DV table for '**Node x**'. One row of the table is pre-populated for you.

Note: The actual exam will ask you to populate the DV table of a different node. The network's link weights may be different.

Can someone tell me which table is correct? And for the bottom table, there are several values.

Commented [207]: Does this mean that there are collisions at the switch? Or no?

Commented [208]: based on what I read in the textbook, it means that the switch explicitly prevents collisions between the hubs, but WITHIN the hubs, there are collisions as they are in the same collision domain. Thus, N3, and N5 can and will collide, but none of the others will collide.

Commented [209]: So does that mean that 1, 4, (3, 5) will collide but they will be resolved?

Commented [210]: yes, the switch will serialize and order the packets to prevent collisions

Commented [211]: How to read the table (for those that are confused like I just was):

we start at the x-node and based on the immediate neighbor column, start with that node as the second node on a route to the destination node on the left column.

ex with given row: for column v, row u, you start at node x, go to node v cause it is the immediate neighbor, then end at node u because it is the destination. thats why it adds to 6!

Commented [212]: I think the first table!

Commented [213]: as in the one directly below us?

so what is the DV algorithm? Is it that at each node, assuming you have gone through the first mandatory neighbor (v,w,y), you do the following:

Check if any neighbor is equal to your target.
if not, pick the lowest-cost neighbor.
Repeat?

That would explain the first row, for the path xvt,

Commented [214]: https://docs.google.com/document/d/14eaIPD8eP5wRoFe8nNUZQ-1NEDVZpwHbrXobi32gaBg/edit?disco=AAAAYtMypCk

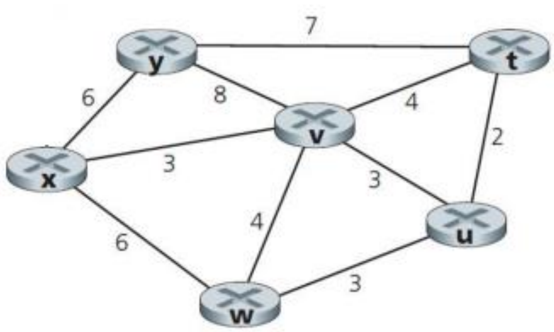
Commented [215]: That is supposed to link the comment, basically just the comment a bit below starting with "Why is this not 7 (xvt)?"

Commented [216]: so the first upper table is correct?

A clear explanation of the algorithm would be helpful.

If the total counts are the same, all paths are available. It is not the matter of path but count

	Cost through immediate neighbors		
Destination	**v**	**w**	**y**
t	7(xvt)	11(xwut)	13(xyt)
u	6(xvu)	9(xwu)	15(xytu)
v	3(xv)	10(xwv)	14(xyv)
w	7(xvw)	6(xw)	18(xyxw)
y	11(xvy)	18(xwvy)	6(xy)



Commented [217]: why not xyvw?

Commented [218]: because we HAVE to go through y in this column.

Commented [219]: yeah XYVW would still go through y in this case

Commented [220]: same result, just a matter of choice

Commented [221]: So we can go back as long as it is the shortest path?

Commented [222]: I got xwuty for this cell but otherwise the exact same answers

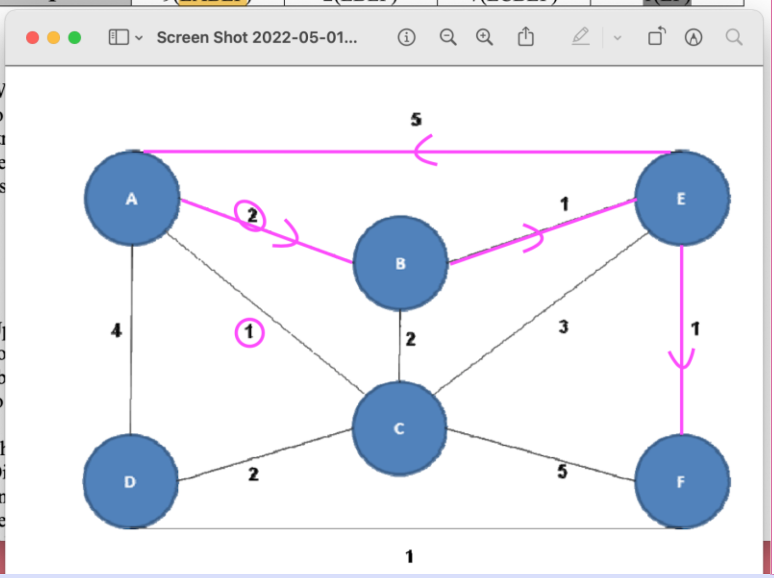
Commented [223]: I think both should be valid

Commented [224]: XWXY = 18, XWVY = 18, XWUTY = 18

Commented [225]: are they all right or is only one of them right though?

Commented [226]: All of them are right. Any path with the minimum cost is a valid answer

	Cost through immediate neighbors			
Destination	A	B	C	F
A	5(EA)	3(BA)	4(ECA)	5(EFDCA)
B	7(EAB)	1(EB)	5(ECB)	6(EFDCB
C	6(EAC)	3(EBC)	3(EC)	4(EFDC)
D	8(EACD)	4(EBEFD)	5(ECD)	2(EFD)
F	9(EABEF)	2(EBEF)	7(ECBEF)	1(EF)



The highlighted cell (in orange) from E through A to F uses EABEF. As shown above, path AB (with cost 2) is chosen rather than AC (with cost 1). This contradicts Sidharta's logic in the comment.

Commented [227]: Where in the textbook can I find this?

Commented [228]: 13-34 13-37

Commented [229]: Fair enough, but that just leaves me more confused on what DV actually does? Is it just the smallest path from src to dst given one neighbor must first be traveled to?

We are effectively shifting the operation of smallest path by 1 node, no?

Commented [230]: Yeah, I think so. It's finding the smallest path from src to dst given one neighbor.

b) (4 points) A transport protocol adds a packet header consisting of the following fields to each packet.

Assume that each of the header fields in the transport layer occupies 4 bytes.

- destination_port
- source_port
- protocol_type

- num_packets
- sequence_number
- packet_size
- check_sum

A network layer protocol adds a datagram header consisting of the following two fields to each packet:

- ipv6 destination_IPaddress
- ipv6 source_IPaddress

A link-layer protocol adds a frame header consisting of the following two fields to each datagram:

- source_MAC_address
- destination_MAC_address

Each IPv6 address is 16 bytes and each MAC address is 6 bytes.

The link-layer operates on a Malav's link layer implementation that can transmit a total of 1590 bytes per frame.

i) Compute the maximum payload packet on the wire. (2 points)

1590 - (7 * 4) - (2 * 16) - (2 * 6) = 1518 bytes

Commented [231]: could someone explain where they found these numbers?

Commented [232]: 7 fields of 4 bytes, 2 fields of 16 bytes, 2 fields of 6 bytes

ii) Yesha wants to send a 94.875KB Word document to Andrej via the internet. Assume that the protocol stack mentioned above minimizes the total number of packets that need to be sent. Assume the network is error free (no collision, no loss of packets, no packet corruption). Compute the number of packets required to send the Word document to Andrej. Show your work for credit. (2 points)

94.875KB / 1518 bytes = 64 packets
(remember that 1 KB = 1024 bytes)

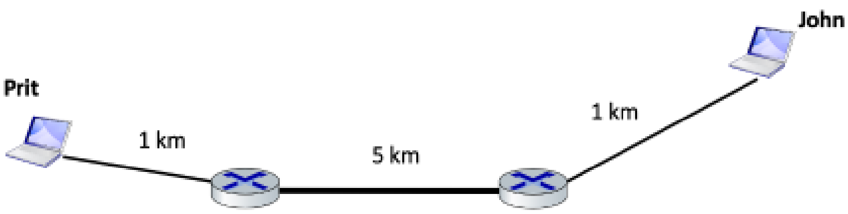
(if the actual exam has different numbers, remember to round up)

Commented [233]: thank you kind soul

c) (6 points) Assume Prit wants to send a message consisting of 1000 packets to John using Malav's 'Sliding Window' reliable transport layer protocol implementation with a window size of 10 packets.

Assume that the network connecting Prit's and John's computers has the following characteristics: ● Transmission speed on the physical link = 2.5 * 10^8 meters/sec

● Assume **each router hop** adds a fixed **cumulative** routing plus queuing latency of 0.1 milliseconds.



● Packet loss = 20% (only for data packets; zero loss for ACKs)
● Sender overhead = 1 millisecond
● Receiver overhead = 2 milliseconds

Assume that the time to place the data packet and ACK on the wire are negligible compared to the propagation time on the medium.

What's the total elapsed time (in milliseconds) for Prit to be certain that the file has been successfully transmitted to John?

The answer for this question is 432 for sure.

T = S + Tw + Tf + R

S = 1, R = 2, Tw = 0, Tf = 0.028 + 0.2 (delay at each router) = 3.228 ms to receive 10 packets

0.228 ms to receive the ack of the first window

3.228 + 0.228 = 3.456 ms

However, since packet loss is 20%, we will have to send 2 additional packets with a time of again 3.456ms

Therefore, in order to get the first 10 packets to the receiver, this will take 3.456 * 2 = 6.912 ms

Then, we need to send the other 900 packets. We get a total time of 6.912 * 100 = 691.2 ms

// I think RTT = 3.456 ms based on the above calculation.

Commented [234]: do we round up or down for loss?

Commented [235]: I think round down, because rounding up would eventually have us infinitely losing 1 packet every time.

Commented [236]: This is per 10 packet window, right? Not per single packet?

Commented [237]: It should be per packet based on the textbook question 13-70

Commented [238]: Imo, "overhead" means all the delays summed together

Commented [239]: TA confirmed it is the same thing as S and R in the textbook. So, there must be 1ms latency per packet if we follow on 13-70

Commented [240]: does that mean that the total time has to be above 2500 ms because of how slow the receiver is?

Commented [241]: Is this the line where we find out that transmission delay for each packet, Tw, equals 0 (negligible)?

Commented [242]: Why 0.028?

Commented [243]: nvm that's 7000m / (2.5*10^8 m/sec)

Commented [244]: why not times 2 to account for return distance as well?

Commented [245]: also i keep getting 5.6 e -5? not .028?

Commented [246]: multiply by 1000 to convert to ms

Commented [247]: just to make sure, this is .1 * 2 because we have two routers along the path from Prit to John right? thank you

Commented [248]: on what basis?

Commented [249]: Do we really need this since its negligible?

Commented [250]: why do we not need to account for sender and receiver overhead for ack packages?

Commented [251]: Because an acknowledgement is a packet in itself. So it's transmission is treated the san ...

Commented [252]: It's a sliding window, so I'm pretty ...

Commented [253]: yes.

Commented [254]: Don't we need the 3.456 to ...

Commented [255]: 13.6.3.2 in text

Commented [256]: I was thinking the same thing. ...

Commented [257]: *90 or *100?

Commented [258]: 100

Commented [259]: At the end, don't we have to add ...

because if packet loss is 20%, the total packets sent = 1000 + 200 + 40

+ 8 + 2 = 1250

total time = 3.456 ms * (1250 / 10) = 432 ms < because a window size is 10. The RTT is the time to spend for sending 10 packets in this case.

I think this ^ answer (calculating total packets first) makes more sense after reading the sliding window section in textbook 13-25

Time to vote:

691.2 ms ➡ this makes more sense tbh (thats)

|||||

432 ms =>

||||||||||||||||

441 ms => makes more sense (See Chapter 13 Example 12 in Textbook, 13-70)

|||

I got something different from both answers above: T = S + Tw + Tf + R

T = 1 ms + 0 ms + ((7000/2.5e8) * 1000 + 2 * 0.1) ms + 2 ms = 1 + 0.228 + 2 = 3.228 ms

T is the time to send a packet but we also need to take into account the sending of an ACK packet, which is the same since Tw is the same.

RTT = 2 * 3.228 ms = 6.456 ms

This is the time to send 10 packets but only receive 1 ACK. The total number of packets we have to send is 1250 packets.

1250/10 = 125 windows

125 * 6.456 ms = 807 ms

At this point, we are still waiting for 9 ACKs, which should each be received a certain time Tr apart.

Commented [260]: how did you get these three numbers?

Commented [261]: We need to send 200 additional packets to account for the lost ones. But when we send 200 of these packets, 20% of them will still be lost. So now we need to send and additional 40 then an additional 40*0.2 = 8, then an additional 8*0.2=(approximately) 2.

Commented [262]: I think the packets sent could also be 1249 (1000 + 200 + 40 + 8 + 1) since it never explicitly said if you should round up or down if you lose packets

Commented [263]: I also agree it is 1249 but when we calculate the total time I guess it'll be same since 124.9 should be rounded up to 125 for the total time..?

Commented [264]: Why divide total packets by 10?

Commented [265]: ^

Commented [266]: 3.456 ms is the amount of time it takes to send 10 packets

Commented [267]: This would make some sense if we were like sending the entire thing at once in a pipeline, but we are only sending them 10 at a time, so wouldn't it make sense to handle per window?

Commented [268]: I agree with you but there aren't any examples in the textbook of this. Also, because the reciever only sends an ACK for the highgest number packet it received, we can't really compute it per window because we can't always assume the first 8 were transmitted correctly and the last two were corrupted. E.g. if the 2nd packet in the window wasn't tramistted, the entire window (from 2 - 10) would have to be resent on the next iteration

Commented [269]: I agree as well. I got 126 windows (instead of 125), which made my answer a bit different

Commented [270]: don't put this here as it will confuse others. make another tally for this instead.

Commented [271]: I will say that the example you site doesn't support your point. the plus nine would lead to an assumption of non parallelism on the delay time, but we are treating them as S and R values, which are concurrent. That plus .45 s which you see in the problem you refer to is referencing a time which is zero in our given problem.

Commented [272]: this doesn't make sense. In sliding window, we send window 2 the moment the ack of the first window is sent. After that, it is a net 1 packet per unit time, no times 2 delay, right?

Commented [273]: I agree, I don't think it should be times 2

Tr = (1 + 2) * 2 = 6 ms (The 1 is accounting for separation of S while 2 is for separation of R. We multiply by 2 since we are sending 2 things for each packet: the packet itself and ACK)

6 * 9 ACKs = 54 ms

807 + 54 = 861 ms, if anyone wants to point out something wrong in my math, please do so

***NOTE: I just realized that sender overhead and receiver overhead could possibly not be happening concurrently which would drastically change the final answer, so I posted a question in piazza, waiting for an answer.

They just posted clarification for this question regarding ACKs

"An additional assumption is added to *simplify* the answer to the question: "The receiver sends an ACK for every packet it receives. Packets are never reordered in this network and the sender selectively retransmits packets it has not received an ACK for. Sending and receiving ACKs does not incur the aforementioned sender/receiver overhead."

The thing that confuses me

Different answers to Q7:

Number of packets to send = 1250 packets (1000 + 200 + 40 + 8 + 2)

Time of flight (Tf) = 0.028ms + 0.1ms + 0.1ms = 0.228ms2

Source side latency per packet = S + Tw = 1 + 0 = 1ms

End to end latency for a DATA packet = S + Tw + Tf + R = 1ms + 0.228ms + 2ms = 3.228ms

End to end latency for an ACK packet = Tf = 0.228ms (since ACK does not incur sender/receiver delay)

Commented [274]: I don't think we need the * 2 because , at this point, the data has been sent, and we are only waiting on ACKs now. At least from the way the textbook did it on 13-70

Commented [275]: does this mean that we don't try to restore the order that packets are sent in? If so, that would put a lot more favor on the 432 ms answer, as that would make a lot more intuitive sense.

Commented [276]: it takes 3.456 ms to send a window of packets. Once u send that window 20% fail so u need to send the extra 2 packets thats another 3.456 ms so 6.912 ms per window. u cant send the next 10 until the previous window succeeded is how i understand it

Commented [277]: it says that packets are never reordered. Does that mean if we sent packets labeled 1,2,3,4,5 and 3 got lost, we resend it, does that give us 1,2,4,5,3 at the receiver end?

Or does never reordered mean that they never break the order they were sent?

Commented [278]: the overhead at each end will ensure the packets remain in order at each end (i think)

Commented [279]: so at the receiver end itll be 1,2,corrupted,4,5 until 3 comes in

Commented [280]: overhead serves to help keep track of lost stuff and ensure sending order is preserved. is that correct?

Commented [281]: I believe this means that 691.2 is the correct answer because for every 10 packets (1 window) 2 fail so u have to retransmit those 2 specifically before you can slide the window (so 6.912 ms per window * 1000/10 windows)

Commented [282]: The thing that confuses me, is that the overhead for sender and receiver are definitely longer than Tf. This means we have to take into account the overheads for each packet in the window. The example in the book(13-71), has a Tf much larger than Tw, S, or R, which is why we did not have to take it into account in the book example, plus S and R were 0, here, they're not.

Commented [283]: i feel like the overhead is the amount of time to prepare the packets of that window but that wouldnt make sense if sometimes we are sending only 2 packets instead of 10 so u might be ...

Commented [284]: S is defined as this in the textbook "This is the cumulative time spent at the sender in the various layers of the protocol stack and includes ...

Commented [285]: let me know if this make sense. I refered to textbook question on 13-70

Commented [286]: why the last remaining ACK is 9?

Commented [287]: why doesn't it?

Commented [288]: read piazza clarification

The first ACK packet received 3.456ms (End to end latency for DATA + End to end latency for ACK) after the first packet was processed (in other words, 2.456ms after the first packet is placed on the wire)

To send 1250 packets, we need 125 cycles (since window size = 10). Therefore, it takes 125 * 3.456ms = 432ms to send all 1250 DATA packets.

The remaining 9 ACK will arrive with a spacing of 1ms (due to 1ms of source side latency for DATA)

Total time to accomplish the message delivery = time for the 1250 DATA packets + time to receive the remaining 9 ACK = 432ms + 9ms = **441**ms

I think this is the most reasonable, but are you sure that the last remaining ACK is 9? If you referred from 13-71, that case is no packet loss.

Sidharta explanation…

The run time from Prit to John: 2 + 0.1 + 0.1 + 0.028 + 1 = 3.228 ms

On average, 2 packets are lost every window. So, we need to send two packets out. Since these two packets can be anywhere from contiguous to opposite ends of the original window, when we decide to selectively retransmit an additional 3.228 ms to send. We repeat this 100 times for 100 windows of 10. So that is 100 * 2 * 3.228 = 645.6 ms. Then, because we are no longer sending any new packets when we start receiving acks for the final window, we are receiving 9 ACKs, each at most separated by 1 ms, so that means at worst 9 ms added, so **654.6 ms**. Does this make sense? This uses the **Selective Repeat method**.

Do we have 1250 or 1249 packets? I thought you're supposed to round down the number of packet losses, so it would be 1000 + 200 + 40 + 8 + 1 instead of 2

^ I don't think that really matters unless they specifically ask us because the number of windows needed is the same(125) in both cases.

I think it would affect the number of remaining ACKS (make it 8 instead of 9)

FINAL ANSWER BELOW=450ms <-likely wrong

Total number of packets = 1000 + (0.2 * 1000) + (0.2^2 * 1000) + (0.2^3*1000)+(0.2^4*1000)=1250

Commented [289]: Why do we not need to add this for all the windows and just the last? the 9ms?

Commented [290]: I also think this is reasonable that reflects both the sliding window and the remaining ACK which are important

Commented [291]: I also think the latency is 3 ms between the ACKs provided that S and R overhead can not happen concurrently, but I'm not sure about concurrency.

Commented [292]: not necessarily. 1 ms between sends, so at worst, acks can be separated by 1 ms, taking 0.228 s to send.

Commented [293]: What are you going to put Sidharta, 432 or 441

Commented [294]: But the Piazza post says that the ACKs don't incur the sender/receiver overheads

End to end latency for a data packet = total distance / speed + (num routers * router hop delay) + source delay + receiver delay

$$= 7km \times \frac{1000m}{1km} \times \frac{s}{2.5\times10^{8}m} \times \frac{1000ms}{1s} + (2 * 0.1) + 1 + 2$$

= 3.228ms

End to end latency for a ack packet = total distance / speed + (num routers * router hop delay)

$$= 7km \times \frac{1000m}{1km} \times \frac{s}{2.5\times10^{8}m} \times \frac{1000ms}{1s} + (2 * 0.1)$$

= 0.228ms

In a duty cycle of 3.228 +0.228ms = 3.456ms 10 data packets have been sent and 1 ack has been received at the source.

To send 1250 packets we need 125 such duty cycles. Therefore the time to send all 1250 data packets = 3.456*125 = 432 ms.

After this time 1250 data packets have been sent and 1241 acknowledgments have been sent. The remaining 9 acknowledgments come in the same gap as the data being sent.

The data is being received at a gap = 2ms. So 432 + (9*2) = 450ms.

???

- I decent on this answer. I put 432, the logic of adding 9 * 2 at the end assumes non parallelism of the ack messages. In the example cited the delay between the sent items is due to the bandwidth of the wire and the time necessary to place the message on the wire. In our case that time is zero, as noted by the last statement where it says "negligible", therefore we just take 432 as all the packets in each window is sent at the same time, and the ack is received 3.456 ms later. Multiply this by 125 to get 432.

Commented [295]: isn't the 2ms overhead not counted for ack sends? the 3.456(125)=432 includes the time for both the send and the acks

Question 8: (Hidden)
Networking protocol stack?
^^Asking about what part of the stack a specific thing is (from HW10 Q6)
ie. Ethernet is a part of which piece of the networking protocol stack?

Something from HW11

Question 9: (Hidden)
Any ideas?
I/O stuff like DMA or PIO
SSDs?
Invalidate or update TLB with multithreaded programs (per processor caching)

IP NETWORKS? - breaking down IP addresses

Maybe SMP Cache Coherence like from Q4 HW9: Chapter 12 pg 12-59 shows another example.

Seems kinda simple but since we had a definition question on the last exam, these topics may be potential hidden questions

- Sequential vs parallel program (which one is deterministic, which one is not deterministic, and what does it mean to be deterministic/nondeterministic)

Other definitions from chapter 12

Concept	Definition and/or Use
Top level procedure	The starting point for execution of a thread of a parallel program
Program order	This is the execution model for a sequential program that combines the textual order of the program together with the program logic (conditional statements, loops, procedures, etc.) enforced by the intended semantics of the programmer.
Execution model for a parallel program	The execution model for a parallel program preserves the program order for individual threads, but allows arbitrary interleaving of the individual instructions of the different threads.
Deterministic execution	Every run of a given program results in the same output for a given set of inputs. The execution model presented to a sequential program has this property.
Non-deterministic execution	Different runs of the same program for the same set of inputs could result in different outputs. The execution model presented to a parallel program has this property.
Data race	Multiple threads of the same program are simultaneously accessing an arbitrary shared variable without any synchronization, with at least one of the accesses being a write to the variable.
Mutual exclusion	Signifies a requirement to ensure that threads of the same program execute serially (i.e., not concurrently). This requirement needs to be satisfied in order to avoid data races in a parallel program.
Critical section	A region of a program wherein the activities of the threads are serialized to ensure mutual exclusion.
Blocked	Signifies the state of a thread in which it is simply waiting in a queue for some condition to be satisfied to make it runnable.
Busy waiting	Signifies the state of a thread in which it is continuously checking for a condition to be satisfied before it can proceed further in its execution.
Deadlock	One or more threads of the same program are blocked awaiting a condition that will never be satisfied.
Livelock	One or more threads of the same program are busy-waiting for a condition that will never be satisfied.
Rendezvous	Multiple threads of a parallel program use this mechanism to coordinate their activities. The most general kind of rendezvous is barrier synchronization. A special case of rendezvous is the thread_join call.