Q1 Networking

Part A

Imagine you are in charge of a company-wide data transfer project where your company needs to transfer 100TB of data stored in the NYC datacenter to the LA datacenter. Come up with two proposals, where one approach has bigger bandwidth but higher latency, and the other approach has lower latency but smaller bandwidth. Describe each proposal in one or two sentences. (Note: You cannot simply use one proposal with different latency/bandwidth assumptions)

Sample answer:

1) send the data wirelessly over the internet: low latency but small bandwidth

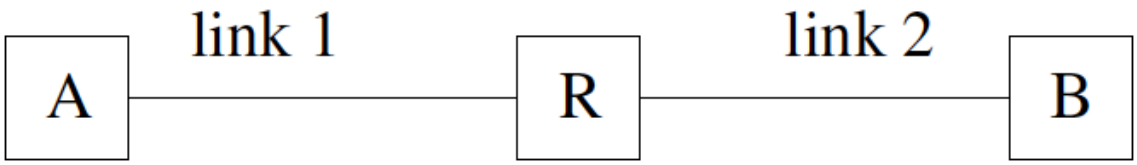
2) store the data in hard disks and hire truck drivers to transport it by highways: large bandwidth but high latency

Part B

For each of the scenarios, state whether you should use UDP or TCP. Briefly explain why.

1. Streaming live 15-440/640 lectures
    a. UDP. It is okay to drop some packets, but we want to make sure that latency is low.
2. A remote SSH session on AFS to test your P0 code
    a. TCP. We want to have a stable session that ensures the delivery of commands.
3. A time server broadcasting the correct time to all machines on a LAN
    a. UDP. TCP is only for two endpoints.

Part C



Alice (A) is trying to communicate with Bob (B) via a network with one router (R) in between. Data from A to B needs to be broken into 10Kbit packets. Alice knows that the link 1 between A and R has 200Mbps capacity, but she doesn't know that of link 2. Alice wanted to test and find out the capacity of link 2.

Test 1: Alice sent a very small piece of data (negligible size) to Bob and had Bob send it back immediately, and she recorded that the time interval is 0.5s.

Test 2: Alice sent over a 1GB file and had Bob send back a checksum of the file immediately, and she recorded that this time interval is 50.0s.
Assume that computing checksum on Bob's computer has a speed of 10Gbps, and Bob can only start computing the checksum after the entire file is received. Ignore all other processing times on the router and Bob.

For all math in the following questions you must show your work to receive full credit. (Reminder: 1B = 8 bits)

1. Assume that the router operates in cut-through mode. What is the capacity of link 2 (in Mbps, keep two decimal points)?
    a. Transmission delay + RTT + processing delay = total delay
    8Gb / minCapacity + 0.5s + 8Gb/10Gbps = 50.0s
    minCapacity = 164.27 Mbps
    This is less than 200Mbps, so link 1 is the bottleneck with 164.27 Mbps.

2. If the router is reconfigured to be in store-and-forward mode, what will the time interval in Test 2 be?
    a. 50.0s + 10Kb / 200Mbps = 50.00005s
    Explanation: the first packet needs to arrive in full at the router before the router can start sending to Bob. Then the router can keep sending the remaining packets without any other delay.

3. The router is configured back to cut-through mode. Assume that connection is lossy and each packet has a 0.001% chance of being corrupted in transit. Assume that any corruption will result in a different checksum. After Test 2, Alice compares the checksum received from Bob with the checksum she has. If they are different, Alice resends the entire file. (For simplicity, ignore the possibility of the checksum being corrupted.) What is the expected number of times that Alice needs to resend the file (round to nearest int)?
    a. # packets = 8Gb / 10 Kb = 800000
    P(file not corrupted) = (1-0.00001) ^ 800000 = 0.000335449
    E(#resends) = 1 / P(file not corrupted) = 2981

    (also accept 2980 if the student did a -1 for counting only "resends")

4. Propose a different protocol where Alice can shorten the expected time that Bob receives the file without corruption. You don't need to show any calculations.
    a. Send back a checksum every ~10000 packets. This means E(#resends for each 10000 packets) = 1 / (1-0.00001) ^ 10000 = 1.105, and expected total time is roughly about 50 * 1.105 + 80 * 0.5 * 1.105 = 100s (this is a very rough estimate)

Note: sending back a checksum each packet is not correct; this would mean at least 800000 * 0.5s = 400000s

Q2 Classic Synchronization

"Office Hours during the pandemic"

Due to the COVID-19 pandemic, the university has published new policies for students attending Office Hours. While a TA can help with exactly one student at a time, other students must wait patiently in a separate waiting room. The waiting room has a maximum capacity of 10 people, and there will only be 10 seats available in the room.
When the TA finishes answering a student's question, he/she dismisses the student and goes to the waiting room. If the TA sees there are other students waiting, he/she will ask one of the students to leave the seat and go to the office, then the TA starts answering the student's question. If there are no students in the waiting room, the TA will return to the office, relaxing for a while until a student comes to the office and asks questions.
When a student arrives, he/she should first check if there are available seats in the waiting room. If the room is full and there are no seats left, the student should leave and attend another OH later. If the student finds a free seat in the waiting room, he/she will grab that seat and then go to the office to check whether the TA is there. If the TA is not there or the TA is busy with answering another student's question, the student will go back to the waiting room, sitting on his/her seat and waiting for the TA to come here. If the TA is taking a rest, the student will release his/her seat in the waiting room and directly start asking questions.

Note: The conditions inside each sub-question below are independent. They only hold in the scope of a sub-question, unless specified otherwise.

1. Tony, a student taking the *Distributed Systems* course this semester, wants to implement the scene above using pseudocode. But there is one thing he disagrees with in the university policy. He believes that TAs and students shouldn't just check each other's availability only on arrival and then wait. When they are waiting, they should both check each other's availability on a frequent basis, for example, every 5 minutes. He starts from drafting some pseudocode of students' and TA's actions:

```
// Student actions
...
lookForTA:
    goToOffice;
    if taAvailable        // If the TA is waiting in the office.
        askQuestion;
```

```
    else
        goToWaitingRoom;
        waitFor5Minutes;
        goto lookForTA;  // Check again after 5 minutes.
                         // Only apply to this sub-question.
...


// TA actions
...
lookForStudent:
    goToWaitingRoom;
    if studentWaiting  // If the TA sees a student waiting in the waiting room.
        answerQuestion;
    else
        goToOffice;
        waitFor5Minutes;
        goto lookForStudent;  // Check again after 5 minutes.
                              // Only apply to this sub-question.
 ...
```

Clearly, without using a synchronization algorithm, the above pseudocode will have many concurrency issues. From what you have learned in class, please explain the concept of a "livelock", and show an example of a potential livelock that might occur in the above scenario.

a. Livelock occurs when two or more processes continually change their states in response to changes in the other processes without doing any useful work.
In the above scenario, assume one student comes to the OH while the TA is just arriving. Simultaneously, the student might go to the office to look for the TA while the TA goes to check the waiting room and find nobody there.
Unfortunately, if the student and the TA repeat their above actions synchronously, the student might wait for an indefinite period of time before his/her question gets answered. This is a potential livelock situation.

2. Some students believe that they could wait outside the waiting room if the room is full. They have proposed a piece of code describing a student's actions:

```
// Global Variables
int numFreeSeats = 10;

// Student actions
void student() {
    onArrive:
        while (numFreeSeats <= 0) {
            // Wait outside the waiting room.
            ;
        }
    // Double-check if there are free seats in the room.
```

```
    if (numFreeSeats > 0) {
        numFreeSeats--;
        waitForMyTurn();  // Block until the student's turn. Assume that it is
                          // implemented correctly with no concurrency issues.
        askQuestion();    // On TA's side, there will be a corresponding:
                          //     numFreeSeats++;
    } else {
        // Wait outside again.
        goto onArrive;
    }
}
```

Assume the TA's actions are implemented correctly. Are there any potential concurrency issues on the students' side in the above code snippet? Please explain your answer.

a. Possible race conditions. The access to the global variable "numFreeSeats" is not synchronized. When there is only one free seat in the waiting room, if two students simultaneously check that (numFreeSeats > 0) holds, and they take a seat at the same time, there will be 11 people in the room. And even worse, the (numFreeSeats--) operation is not guaranteed to be atomic , so there can be more chances of race conditions, resulting in more people in the room.

3. Jane, one of the top students in the class, comes up with a more complex implementation of the behavior of the TA and the students, by using semaphores:

```
// Global Variables
int numFreeSeats = 10;       // The capacity of the waiting room.
semaphore taReady = 0;       // Initialized to 0.
semaphore checkSeats = 1;    // Initialized to 1.
semaphore studentReady = 0;  // Initialized to 0.

// TA actions
void ta() {
    while (true) {
        P(studentReady);
        P(checkSeats);
        numFreeSeats++;
        V(taReady);
        V(checkSeats);
        answerQuestion();  // Block until finish answering a student's
                           // question.
    }
}

// Student actions
void student() {
    P(checkSeats);
    if (numFreeSeats > 0) {
        numFreeSeats--;
        V(studentReady);
```

```
        V(checkSeats);
        P(taReady)
        askQuestion();  // Block until the question is answered.
    }
    // Leave otherwise.
}
```

However, when she tests her code, as more and more students come to the OH, the program blocks at some point. Please identify this concurrency issue, explain why it would happen, and propose a fix to it by making changes to the code. (Hint: You shouldn't add/delete/modify more than 5 lines of code.)

a. Deadlock. In the student actions, the program forgot to add an else branch to signal the "checkSeats" semaphore to yield it to others when there are no free seats. So if a student arrives and leaves after the waiting room is full, anyone waiting on "P(checkSeats)" later will get blocked forever, and the program is deadlocked.
   The fix to it is to add an else branch after the if statement in student actions:

```
// Student actions
void student() {
    P(checkSeats);
    if (numFreeSeats > 0) {
        numFreeSeats--;
        V(studentReady);
        V(checkSeats);
        P(taReady)
        askQuestion();  // Block until the question is answered.
    }
    // Leave otherwise.
    else {
        V(checkSeats);
    }
}
```

4. Consider the code in sub-question 3 with your fix to the bug. Assume there are two students coming to the OH, one after another. The second student arrives while the TA is answering the first student's question. The second student waits for 10 minutes in the waiting room before the TA dismisses the first student and picks up the second student. Please fill in the following table, where each line represents an event of changing in the value of a semaphore. The events are happening in sequential order.

   Note: There will be only one correct answer to receive full points. During the procedure, there can be cases where two events can happen in any order. For those cases, we will specify one possible case and fill in those rows for you.

Answer Table:

Event Index	Action Taker (TA / Student1 / Student2)	checkSeats	taReady	studentReady
Initial	\	1	0	0
1	Student1	0	0	0
2	Student1	0	0	1
3	Student1	1	0	1
4	TA	1	0	0
5	TA	0	0	0
6	TA	0	1	0
7	TA	1	1	0
8	Student1	1	0	0
9	Student2	0	0	0
10	Student2	0	0	1
11	Student2	1	0	1
12	TA	1	0	0
13	TA	0	0	0
14	TA	0	1	0
15	TA	1	1	0
16	Student2	1	0	0

5. Consider the code in sub-question 3 with your fix to the bug. Do you see another remaining problem with this implementation? (Hint: Is this implementation fair to every student?) If you identify the problem, briefly explain it and describe a possible solution. (Plain text only, no pseudocode needed.)

    a. Starvation. When there are multiple students in the waiting room, the TA always picks the next student arbitrarily. Some students might wait indefinitely before

their question gets answered. A possible solution is to use a FIFO queue where students are added as they arrive, so that the TA can serve them on a first come first served basis.
There is another acceptable answer to this question: a student may spend an indefinite amount of time asking a question. If the TA is stuck at one student's question, other students will not receive the TA's help. Adding a timeout mechanism can prevent one student from taking forever.

Q3 Time Synchronization

Consider three processes. The system has totally ordered clocks by breaking ties by process ID. It uses the Ricart & Agrawala algorithm. The timestamp for each process of ID i is $T(p) = 10 * L(p) + i$, where $L(p)$ is a regular Lamport clock.

Each message takes 2 'real time' steps to get delivered. The critical section takes 4 real-time steps. This means that if a process receives all replies at real time t-1, it starts executing the critical section at time t, and finishes executing and sends messages at time t+4.

Assume that if a process receives messages from the other two processes at the same time, the message that comes from the lower process ID will be received first.

Fill in the table with the messages that are being broadcast, sent, or received between the processes until all nodes have executed their critical sections. The first few rows have been filled for you. Then use your filled table to answer the questions below.

Action types: Broadcast (B), Receive (R), Send (S), Execute Critical Section (ExecCS), Exit Critical Section (ExitCS).
Initial timestamps:
L(p1) = 19, T(p1) =191,
L(p2) = 27, T(p2) = 272,
L(p3) = 6, T(p3) = 63

Real Time	Process ID	Lamport Time	Action	Message	Queue at P1	Queue at P2	Queue at P3
1	1 3	201 73	B B	request 201 request 73	201		73
2	2	282	B	request 282	201	282	73

3	1 2 2 3	211 292 302 213	R from 3 R from 1 R from 3 R from 1	request 73 request 201 request 73 request 201	73 201	73 201 282	73 201
4	1 1 2 2 3	291 301 312 322 293	R from 2 S to 3 S to 3 S to 1 R from 2	request 282 reply 73 reply 73 reply 201 request 282	201 282	282	73 201 282
5					201 282	282	73 201 282
6	1 3 3	331 313 323	R from 2 R from 1 R from 2	reply 201 reply 73 reply 73	201 282	282	73 201 282
7	3	333(323)	Exec CS		201 282	282	201 282
8					201 282	282	201 282
9					201 282	282	201 282
10					201 282	282	201 282
11	3 3 3	343(323) 353(333) 363(343)	ExitCS S to 1 S to 2	 reply 201 reply 282	201 282	282	
12					201 282	282	
13	1 2	361(341) 372(352)	R from 3 R from 3	reply 201 reply 282	201 282	282	
14	1	371(341)	Exec CS		282	282	
15					282	282	
16					282	282	
17					282	282	

18	1 1	381(341) 391(351)	ExitCS S to 2	reply 282		282	
19						282	
20	2	402(362)	R from 1	reply 282		282	
21	2	412(362)	Exec CS				
22							
23							
24							
25	2	422(362)	Exit CS				

Questions: (answers in brackets are for not counting ExecCS/ExitCS as events that incr Lamport clock)

1. At what real time does each process start executing the critical section?
    a. Process 1: 14
    b. Process 2: 21
    c. Process 3: 7
2. At what timestamp does each process start executing the critical section?
    a. Process 1: 371 (341)
    b. Process 2: 412 (362)
    c. Process 3: 333 (323)
3. What does the queue for each process look like at real time 12?
    a. Process 1: 201, 282
    b. Process 2: 282
    c. Process 3: empty

Q4 Distributed Mutual Exclusion

1. Describe the fairness of each of the following distributed mutual exclusion algorithms: Ricart & Agrawala, token ring, decentralized mutual exclusion, and centralized mutual exclusion. Please use 2 or fewer sentences for each algorithm.
    a. Ricart & Agrawala: fair, requests are granted in the order they are made
    b. Token ring: fair, there will always be $\leq$ n-1 accesses by other nodes before a node gets access again
    c. Decentralized: depends on random chance

    d. Centralized: depends on queueing policy
2. Describe how each of the following distributed mutual exclusion algorithms is impacted by node failure: Ricart & Agrawala, token ring, decentralized mutual exclusion, and centralized mutual exclusion. Please use 2 or fewer sentences for each algorithm.
    a. Ricart & Agrawala: if the node with the CS crashes and there is no time-out function, all other nodes will stall waiting for the CS to be released
    b. Token ring: if a node crashes the token is lost and all other nodes stall waiting for the token
    c. Decentralized: if a node fails before it has voted and does not vote upon reboot it may stall the system if no one can get majority
    d. Centralized: if the master node fails and there is no election procedure the other nodes will stall waiting for permission to enter the CS (an election procedure alleviates this issue), if a node crashes in the CS and does not release access and there is no time-out function the master node will not grant access to any other nodes
3. Explain why the Ricart & Agrawala algorithm is starvation and deadlock free.
    a. Starvation free: if a node makes a request with a timestamp t, eventually all other nodes will update their local clock to be greater than t and therefore the request with time t will eventually be the request with the smallest timestamp
    b. Deadlock free: deadlock would occur if there was a cycle of nodes waiting for each other, however this would require every node in the cycle to have a greater request timestamp than the next which is not possible in a cycle

Q5 Remote Procedure Calls

For each of the following scenarios, state whether you would use an RPC guaranteeing at-most-once, at-least-once, or exactly-once semantics (exactly-once should work in all cases, but if it is overkill you should pick the weakest guarantee that is enough for you application). In some cases multiple answers can be accepted, but you need to make sure your explanation is clear and coherent with your answer, and your (reasonable) assumptions are made explicit. Explain in no more than 3 sentences.

1. Depositing money into a bank account
    a. recommended answer: exactly-once because you wouldn't want the account to get the money twice, but it is important that the account reflect the money that's been handed over
2. Uploading your proof of vaccination to SIO
    a. recommended answer: at-least-once so that you can come to campus, but don't need exactly once because it's fine if it get put in twice

3. Making a login attempt online where your account will be frozen after 3 incorrect attempts
    a. recommended answer: at-most-once, because it's okay if you have to type it in more than once, but getting your account frozen would be bad
4. Cancelling a fraudulent order that's been placed on your credit card
    a. recommended answer: at-least-once, it's okay if the order gets cancelled twice but you definitely don't want to pay for the fraudulent charges
5. Booking a cross-country airplane ticket to go to your internship
    a. recommended answer: exactly-once because it's important to be at work on time but you don't want to pay for an expensive ticket twice