Question 1 (25 points)

A. As the teacher is not present in the room with the student, some complexities arise. For example, while figuring out a long, complex problem on a blackboard, you can’t simply point to a questionable step and ask if it looks right. What must you do instead? What analogous problem arises in RPC systems?

Carefully describe the problem in your message.
The analogous problem is cannot pass by reference, solved by serialization.

B. An owl takes 1 day to fly from your house to Hogwarts, and 1 day to get back. You need to send an owl to get your grade on the midterm in Potions class, check to see if you still have any overdue materials that need to be returned to the library, and ask for a deadline extension on your Charms class homework. However, you do not want to wait 6 days for an owl to make 3 trips back and forth. How might you get your tasks done sooner?

Send 3 owls in parallel. Or, send multiple messages with the same owl (with a new RPC format).

C. You sent a question by owl, but 3 days later, you still have not received a reply. What might have happened? What should you do next?

The owl got lost, died, etc.
Retry by sending a new owl. Mourn your old owl (optional).

D. The Hogwarts accounting department received an owl message from Ron Weasley that directed them to transfer 2 gold coins from his account into yours. They do so, and send Ron a message indicating the transfer was completed. A few days later, they receive an identical message to transfer 2 gold coins. Why might this happen? Should they transfer 2 coins again? How might Ron have made the situation clearer?

Ron sent multiple messages to ensure at least one was delivered. Or, a third party intercepted and re-sent a copy of message.
The bank should not transfer 2 coins again.
Ron can include a nonce / sequence number so that the bank can differentiate duplicate messages from messages which happen to have the same content.

E. Borrowing books from the library is a big problem, as most books are too heavy for owls to carry. So, the library magically copies, and splits books into 20-page sections that can be comfortably carried by an owl. The borrower needs to reassemble the book from the many sections. Suppose you wish to borrow a 315-page book. How long will it take to
get the book if the library is willing to devote 5 owls to the task? (Remember it takes 1 day for an owl to fly the distance between Hogwarts and your house).

Assume all owls are successful. There are 315 / 20 = 16 sections.

Day 1: send an owl to the library requesting the book
Day 2: 5 owls fly with pages 1-100 to your house.
Day 3: 5 owls return to Hogwarts.
Day 4: 5 owls fly with pages 101-200 to your house.
Day 5: 5 owls return to Hogwarts.
Day 6: 5 owls fly with pages 201-300 to your house.
Day 7: 5 owls return to Hogwarts.
Day 8: 1 owl flies to your house with pages 301-315.

= 8 days after the first set of owls is dispatched.
(9 is incorrect, the last owl does not need to return to Hogwarts for you to have finished receiving your book, but give partial credit if work is shown).
Question 2 (20 points)

CMU is testing out a new classroom management system called Pizza. Rather than using a boring Internet-browser-based interface, Pizza provides a cool desktop application for students to ask questions and submit assignments. Likewise, it provides a different desktop application for TAs, which has a beautiful GUI customized for hand-grading submissions and answering questions. The desktop applications do not directly communicate with each other; instead, a secure central computer in Farnam’s basement manages all interactions between student and TA applications through a set of RPCs.

A. Consider the following RPC used by Farnam’s computer to ask a particular TA to grade a student submission and update the gradebook:

```c
void assign_grade ( pdf_t submission,
    char *andrew_id,
    gradebook_t *gradebook,
    int32 *grade );
```

Here, `pdf_t submission` is the actual pdf of the student’s work, `char *andrew_id` is the student’s andrew id, `gradebook_t *gradebook` is the complete gradebook for the very large class, and `int32 *grade` is the computed grade of the student for this assignment.

   a. For this RPC call, which machine is acting as a server and which is acting as a client?

   **Accepted Solution:**

   Client: Farnam’s Computer

   Server: TA’s Computer

   b. What are the **in**, **out**, and **in-out** parameters of this RPC?

   **Accepted Solution:**

   In: submission, andrew_id

   Out: grade

   In-Out: gradebook

B. Your favorite 440 TA thinks the RPC described in part (A) is poorly-designed for this application. Why might she/he feel this way? Identify and briefly explain three issues
with the RPC. (Hint: think about complexities in the RPC stub code, and duration of the RPC call; also, remember this is a large class with many students and TAs)

Accepted Solutions

● A shared gradebook serializes grading or causes races / write conflicts
● Gradebook is a large data structure that doesn't need to be exchanged across the network.
● This function call requires a human in the loop. How would the client distinguish between a network timeout and a user taking a long time to grade?
● You can't exchange pointers between applications
● Void return type makes it difficult to determine if there was an rpc failure.
● Batching submissions as an improvement

C. Think about what operations would be required in Pizza to allow students to post questions, and TAs to answer them. Briefly describe a set of RPC interfaces that you have designed to facilitate this. You should include a prototype of each RPC, including the name and parameters of each, as well as a brief explanation of what the RPC does.

Example Solution

typedef struct user
{
    char *andrew_id;
    char *authentication_token;
} user_t;

typedef struct question
{
    int id;
    char *andrew_id;
    int time_stamp;
    char *question;
    char *answer;
} question_t;

/**
 * Authenticates a user
 */
int authenticate_user(char *andrew_id, char *password, user_t **me);

int post_question(char *question, user_t *student);

int get_students_questions(user_t *student, question_t **my_questions, int n);

int get_unanswered_questions(user_t *ta, question_t **questions, int n);
/*
 * Answers a question
 * @in-param question a student's question
 * @in-param answer the ta's answer
 * @in-param ta the current authenticated ta
 * @returns 0 on success and negative on network failure
 */

int get_unanswered_questions(question_t **questions, int *n, user_t *ta);

int answer_question(question_t *question, char *answer, user_t *ta);
Question 3 (19 points)

A. Briefly describe the 3 different RPC semantics discussed in lecture and the associated pros/cons with them?

Sample Answer: (Note: Any reasonable explanation for pros/cons can be accepted)

- At-least-once - Ensures that the call gets executed at least once (>= 1 times)
  - Pro: Simple to implement, implementation can be stateless on server side
  - Con: Requires operation idempotency
- At-most-once - Ensures that the call gets executed at most once (either 0 or 1 time)
  - Pro: Do not need to keep track of sequence numbers.
  - Con: Server may hold resources, slowing down other activities
- Exactly-once - Ensures that the call gets executed exactly once (1 time)
  - Pro: Always works, no failure
  - Con: Difficult to implement

B. For each of the following scenarios, select one of the semantics you described in part (A) that best fits the scenario. In some cases, multiple answers may be accepted, but you need to make sure your explanation is clear and coherent with your answer, and your (reasonable) assumptions are made explicit. Note: You should always strive to use the weakest semantics possible (we know that there is one that can always be used :))

Explain in no more than 2 sentences.

a. Submitting a problem set to Gradescope. Note: You have unlimited submissions.

Sample Answer: In this situation, you should use at-least-once semantics. Since we have unlimited submissions, there is no issue if we accidentally resubmit.

b. Submitting P1 to Autolab. Note: You have limited submissions.

Sample Answer: In this situation, you should use at-most-once semantics. Since we have limited submissions, we do not want to accidentally submit multiple times and lose one of our precious unpenalized submissions. In addition, we can directly check the submission result soon after we send the request, unlike a bank transaction which may take a few days to process.

Exactly-once semantics could work but would not be preferred. Since Autolab can potentially have lots of students at the same time, exactly-once may have a potentially high cost.

c. Sending a friend request on Facebook.

Sample Answer: In this situation, you should use at-least-once semantics. Even if we send a
friend request multiple times, it makes no difference since the user will only receive 1 friend request from us.

d. Downloading the entire series of the TV show *Friends* (all 10 seasons).

Sample Answer: In this situation, you should use at-most-once semantics. If we repeatedly try to download the entire series, we could potentially put our computer’s CPU at risk since it would be overloading the CPU and could affect other background processes.

e. Paying your tuition for the upcoming semester.

Sample Answer: In this situation, you should use exactly-once semantics. We need to make sure that our payment goes through so that we will not be prevented from enrolling for the upcoming semester. But we also need to make sure that we do not accidentally send multiple 70K payments to the university.

(Some may ask about at-most-once, but due to transaction latency of banks, checking results of payment and repaying if necessary can be costly.)
Question 4 (20 points)

CryptoCom, Inc. encrypts all traffic sent between its servers. Currently, traffic is encrypted on the CPU of the sending server and decrypted on the CPU at the receiver. The CPUs on these servers can encrypt and decrypt at a rate of 1 GB/sec.

Engineers at CryptoCom have a prototype design for a network switch that can perform encryption at decryption at 4 GB/sec. Using this new technology, a sender now sends unencrypted data from its network interface, but the data is routed through the new network switch, which will encrypt it and send the data along its route to the receiver. Similarly, traffic would be decrypted at the last hop network switch, and delivered in unencrypted form to the receiver.

You are tasked with helping to determine whether this new design is a good idea, both quantitatively and qualitatively. For all quantitative questions, you can assume SI units, e.g., 1 KB = 10^3 bytes. You can also assume that encryption of an entire message must complete in full before being sent over the network (i.e., there is no overlap between encryption, decryption, and network transfer).

A. Suppose that this were deployed at CryptoCom for intra-datacenter communication (i.e., that between two servers sitting in the same datacenter). In this setting, network delay is typically 1 ms, network bandwidth is typically 50 Gb/sec (note, Gb as in “gigabit”), and messages are 100 MB in size on average. What speedup in end-to-end transfer latency would deploying the new switches bring to this setup?

Accepted solution 1: This solution follows the instructions in the original question that computation/transfer cannot be overlapped. Due to the 3 network hops in the switch design, one would pay a transmission cost three times.

\[
\begin{align*}
\text{tE2e} & = \text{tEncrypt + tNetworkDelay + tTransfer + tDecrypt} \\
\text{tEncrypt} & = \text{tDecrypt} \\
\text{tCryptCpu} & = \frac{\text{messageSizeBytes}}{\text{encryptCpuBytesPer}} \\
& = \frac{(100 \times 10^6)}{10^9} \\
& = 0.1 \\
\text{tCryptSwitch} & = \frac{\text{messageSizeBytes}}{\text{encryptSwitchBytesPer}} \\
& = \frac{(100 \times 10^6)}{(4 \times 10^9)} \\
& = 0.025 \\
\text{tTransfer} & = \frac{\text{messageSizeBytes}}{\text{networkBytesPer}} \\
& = \frac{(100 \times 10^6)}{(50 \times 10^9 / 8)} \\
& = 0.016 \\
\text{tTransferSwitch} & = \text{numHops} \times \text{tTransfer} \\
& = 3 \times 0.016
\end{align*}
\]
tE2eCpu = tCryptCpu + tNetworkDelay + tTransfer + tCryptCpu
= 0.1 + 0.001 + 0.016 + 0.1
= 0.217 seconds

tE2eSwitch = tCryptSwitch + tNetworkDelay + tTransferSwitch + tCryptSwitch
= 0.025 + 0.001 + 0.048 + 0.025
= 0.099 seconds

Speedup = tE2eCpu / tE2eSwitch
= 0.217 / 0.099
= 2.19

We also accepted the difference between the two (i.e., 0.217 - 0.099 = 0.118), as speedup was not defined in the question.

**Accepted solution 2:** We also accepted the solution wherein transfer time is overlapped, as many students found the working of this portion confusing.

```
tE2e = tEncrypt + tNetworkDelay + tTransfer + tDecrypt

tEncrypt = tDecrypt

tCryptSwitch = messageSizeBytes / encryptSwitchBytesPer
= (100 * 10^6) / (4 * 10^9)
= 0.025

tTransfer = messageSizeBytes / networkBytesPer
= (100 * 10^6) / (50 * 10^9 / 8)
= 0.016

tE2eCpu = 0.217 seconds (same as above)

tE2eSwitch = tCryptSwitch + tNetworkDelay + tTransfer + tCryptSwitch
= 0.025 + 0.001 + 0.016 + 0.025
= 0.067 seconds

Speedup = tE2eCpu / tE2eSwitch
= 0.217 / 0.067
= 3.24
```

We will also accept the difference between the two (i.e., 0.217 - 0.067 = 0.15), as speedup was not defined in the question.

B. **Suppose that this were deployed at CryptoCom for inter-datacenter communication (i.e., that between two servers sitting in different datacenters). In this setting, average network delay is 100 ms, average network bandwidth is 1 Gb/sec (note, Gb as in “gigabit”), and messages are on average 1 KB. What speedup in end-to-end transfer latency would deploying the new switches bring to this setup?**

**Accepted solution 1:** This solution follows the instructions in the original question that
computation/transfer cannot be overlapped. Due to the 3 network hops in the switch design, one would pay a transmission cost three times.

\[
\begin{align*}
\text{tE2e} &= \text{tEncrypt} + \text{tNetworkDelay} + \text{tTransfer} + \text{tDecrypt} \\
\text{tEncrypt} &= \text{tDecrypt} \\
\text{tCryptCpu} &= \frac{\text{messageSizeBytes}}{\text{encryptCpuBytesPer}} \\
&= \frac{10^3}{10^9} \\
&= 0.000001 \\
\text{tCryptSwitch} &= \frac{\text{messageSizeBytes}}{\text{encryptSwitchBytesPer}} \\
&= \frac{10^3}{(4 \times 10^9)} \\
&= 0.00000025 \\
\text{tTransfer} &= \frac{\text{messageSizeBytes}}{\text{networkBytesPer}} \\
&= \frac{10^3}{(10^9 / 8)} \\
&= 0.000008 \\
\text{tTransferSwitch} &= \text{numHops} \times \text{tTransfer} \\
&= 3 \times 0.000008 \\
&= 0.000024 \\
\text{tE2eCpu} &= \text{tCryptCpu} + \text{tNetworkDelay} + \text{tTransfer} + \text{tCryptCpu} \\
&= 0.000001 + 0.000008 + 0.1 + 0.000001 \\
&= 0.10001 \text{ seconds} \\
\text{tE2eSwitch} &= \text{tCryptSwitch} + \text{tNetworkDelay} + \text{tTransferSwitch} + \text{tCryptSwitch} \\
&= 0.00000025 + 0.000024 + 0.1 + 0.00000025 \\
&= 0.1000245 \text{ seconds} \\
\text{Speedup} &= \frac{\text{tE2eCpu}}{\text{tE2eSwitch}} \\
&= \frac{0.10001}{0.1000245} \\
&= 0.9999 \\
\end{align*}
\]

We also accepted the difference between the two (i.e., \(0.10001 - 0.1000245 = -0.0000145\)), as speedup was not defined in the question.

**Accepted solution 2:** We also accepted the solution wherein transfer time is overlapped, as many students found the working of this portion confusing.

\[
\begin{align*}
\text{tE2e} &= \text{tEncrypt} + \text{tNetworkDelay} + \text{tTransfer} + \text{tDecrypt} \\
\text{tEncrypt} &= \text{tDecrypt} \\
\text{tCryptCpu} &= \frac{\text{messageSizeBytes}}{\text{encryptCpuBytesPer}} \\
&= \frac{10^3}{10^9} \\
&= 0.000001 \\
\text{tCryptSwitch} &= \frac{\text{messageSizeBytes}}{\text{encryptSwitchBytesPer}} \\
&= \frac{10^3}{(4 \times 10^9)} \\
&= 0.00000025 \\
\text{tTransfer} &= \frac{\text{messageSizeBytes}}{\text{networkBytesPer}} \\
&= \frac{10^3}{(10^9 / 8)}
\end{align*}
\]
\[ t_{E2eCpu} = t_{CryptCpu} + t_{NetworkDelay} + t_{Transfer} + t_{CryptCpu} \]
\[ = 0.000001 + 0.000008 + 0.1 + 0.000001 \]
\[ = 0.10001 \text{ seconds} \]

\[ t_{E2eSwitch} = t_{CryptSwitch} + t_{NetworkDelay} + t_{Transfer} + t_{CryptSwitch} \]
\[ = 0.00000025 + 0.000008 + 0.1 + 0.00000025 \]
\[ = 0.1000085 \text{ seconds} \]

\[ \text{Speedup} = \frac{t_{E2eCpu}}{t_{E2eSwitch}} \]
\[ = \frac{0.10001}{0.1000085} \]
\[ = 1.000014999 \]

We also accepted the difference between the two (i.e., \( 0.10001 - 0.1000085 = 0.0000015 \)), as speedup was not defined in the question.

C. You may have noticed that the speedup achieved when using this technology differed in the settings described in (a) and (b). Explain why this is the case. (Hint: think about what the bottlenecks are in each scenario)

In case (a), the bandwidth of the units performing cryptography was the bottleneck, so improving it by a factor of 4 leads to significant end-to-end latency reductions. In case (b), network delay was the bottleneck, so improving the bandwidth of cryptography provides little benefit.

D. Describe qualitatively why this design would be a good or a bad idea, using arguments from the end-to-end principle.

From the end-to-end principle, this design is a bad idea:

a. It leaves an unencrypted path open between the switches and the client/server, which provides a potential vulnerability

b. It pushes core functionality of the service (end-to-end encryption) into the network. The resulting design may be harder to maintain.
Question 5 (16 points)

Which of the following is a violation of safety properties? Which of the following is a violation of liveness properties? Explain why.

A. The traffic light at an intersection is always green for all 4 directions.
   Safety

B. The traffic light at an intersection is always red for all 4 directions.
   Liveness

C. A shared document only allows one person to edit it at a time and the next person cannot begin to edit it until the previous person closes it.
   Liveness

D. A shared document allows anyone to edit it at a given time.
   Safety