Department of Electrical Engineering and Computer Science

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

6.894 Distributed Operating Systems Engineering: Fall 2000

Final

This final has 13 questions. In order to receive credit you must fill in the blank(s) or mark the correct answer or answers for each question. You have three hours to answer this exam, though we expect that 1.5 hours will be sufficient.

Write your name on this cover sheet AND at the bottom of each page of this booklet.

Some questions may be much harder than others. Read them all through first and attack them in the order that allows you to make the most progress. If you find a question ambiguous, be sure to write down any assumptions you make. Be neat. If we can't understand your answer, we can't give you credit!

THIS IS AN OPEN BOOK, OPEN NOTES FINAL.

Do not write in the boxes below

1-3 (xx/30)	4-5 (xx/20)	6-8 (xx/30)	9-10 (xx/20)	11-13 (xx/30)	Total (xx/130)

Name: Solutions

I Byzantine fault tolerance

Answer the following questions about the paper "Practical Byzantine fault tolerance" by Castro and Liskov.

1. [10 points]: Suppose one faulty BFS replica returns fake results to clients but otherwise faithfully executes the protocol. Will this cause a client to observe incorrect results (e.g. an old value rather than the last written value)? Answer yes or no, and explain your answer briefly.

Answer: No. The client will look for $\geq f + 1$ answers which are the same before it believes in the result, therefore one faulty BFS replica could not get the clients to accept the fake result

2. [10 points]: Why do backups accept pre-prepare messages with sequence numbers between h and H instead of accepting just the next expected sequence number? (Explain briefly).

Answer: This is for performance optimization (similiar to the pipeline principle) Backups accept more than one pre-prepare messages so that out-of-order messages will be accepted and ordered later.

3. [10 points]: This question explores how the protocol handles a primary that skips sequence numbers. Can clients observe incorrect results if a primary sends a pre-prepare message with sequence number 10, completely skips sequence numbers 11 through 19, then sends a message with sequence number 20, but otherwise executes the protocol faithfully? (Assume h = 0 and H = 100). Answer yes or no, and explain your answer briefly.

Answer: No. Clients must wait for requests 11-19 before they execute request 20. Therefore, eventually timeout will happen and the primary will be considered faulty. A view change will result before the protocol proceeds

II SSL 3.0

Consider the SSL handshake protocol between a client (C) and a server (S) as depicted below:

A. C ⇒ S : ClientHello (client_version, random, session_id, cipher_suites)

B. S ⇒ C: ServerHello (server_version, random, session_id, cipher_suites)

C. S ⇒ C: ServerCertificate (certificate_list)

D. S ⇒ C: ServerHelloDone

E. C ⇒ S: ClientKeyExchange($E(\text{pre\_master\_secret})_{\text{ServerPubKey}}$)

F. C ⇒ S: ChangeCipherSpec (cipher_suite)

G. C ⇒ S: [Finished(MAC(master_secret, all handshake messages)]

H. S ⇒ C: ChangeCipherSpec (cipher_suite)

I. S ⇒ C: [Finished, MAC(master_secret, all handshake messages)]

J. C ⇔ S: [Data(plaintext)]

Notes:

- Messages between [] are encrypted and authenticated.
- Steps **A** and **B** may list suites of many ciphers.
- Step **C** transfers a list of the server's certificates, signed by certificate authorities. The client verifies one with the public key of the authority.
- Step **E** typically uses RSA to encrypt the pre_master_secret with the server's public key.
- Step **F** announces which cipher the client will use for the remaining messages.
- The client and server calculate the cipher session keys after steps **F** and **H**.
- Steps **G** and **I** allow the server and client to verify that none of the previous messages were tampered with.

The purpose of the protocol is to prove the server's identity to the client, and to set up an encrypted and authenticated communication channel between client and server.

Remember that the master secret is a function of the random number chosen by the client, the random number chosen by the server, and a pre_master_secret chosen by the client.

Name:

4. [10 points]: What attack would be possible if the protocol were modified to leave out the random number supplied by the server in step **B**, and to have client and server compute the master secret from just the random number in step **A** and the pre_master_secret? Give a concrete example of the attack.

Answer: Replay attack is possible. Specifically, the server could not detect a replay of the handshake with an earlier client by the attacker

5. [10 points]: What attack would be possible if the protocol were modified to omit message **E** and compute the master secret from the random numbers in messages **A** and **B** alone? Give a concrete example of the attack.

Answer: The master secret will be compromised. An eavesdropper could learn about the random numbers in messages A and B (which are sent in clear text) and therefore compute the master secret. Once master secret is compromised, all sessions keys are no longer secret.

Name:

III Bayou

ValuJet decides to implement its own airplane seat reservation system using the Bayou distributed database described in the paper "Flexible Update Propagation for Weakly Consistent Replication," by Petersen, Spreitzer, Terry, Theimer, and Demers. Each flight has an independent seat database, indicating whether each seat is reserved or available. To keep the problem simple, assume there is just one flight and one database. The flight has just ten seats, numbered 1 to 10. ValuJet's central computer runs the Bayou "primary replica" of the database. Each sales agent's laptop keeps a replica database. The laptops are ordinarily not connected to any network, but from time to time the agents can perform anti-entropy exchanges between pairs of laptops or between a laptop and the primary replica. The laptops and primary replica do not have synchronized clocks.

A Bayou "write" in this system consists of a dependency/merge procedure along with the customer's name. When anti-entropy copies writes among replicas, it is the procedure and name that are copied, not the seat assignments. The procedure finds the lowest-numbered open seat and, if one is available, marks it as reserved for the customer. Here is the dependency/merge procedure:

```
merge_procedure(char *customer) {
  int s;
  for(s = 1; s <= 10; s++){
    if(seat s is not reserved){
      reserve seat s for customer;
      return;
    }
  }
  /* there are no seats left */
}
```

An example call in a replica's log could be merge_procedure("Smith"), which would result from an attempt to reserve a seat for customer Smith.

ValuJet has only three sales agents, Q, M, and Felix, and they sit next to each other in the head office. Each agent has a laptop containing a database replica, so there are four replicas in total. Each laptop displays a line of information about each seat number, containing the name of the passenger who has reserved it.

(Turn the page...)

Name:

6. [10 points]: As our scenario opens, no seats are reserved. Then Agent Q reserves a seat for Frans Kaashoek, and then a seat for Robert Morris. Then Agent M reserves a seat for Hari Balakrishnan. Then the two agents perform anti-entropy between their replicas. Their laptops now display the flight's seat assignment; a possible display might be "1: Kaashoek; 2: Morris; 3: Balakrishnan." You can assume that only the actions described above have occurred. Does Bayou (the complete system with all mechanisms) guarantee that both laptops display the same assignments? Answer yes or no, and briefly explain your answer.

Answer: Yes. Since Agent Q and M have performed an anti-entropy with each other (Since anti-entropy protocol is uni-directory, the question really means Q has performed anti-entropy with M and vice versa), both will have the same set of "writes". Since all writes are displayed in an order according to (CSN, accept-stamp,server-ID) tuple (there is no CSN here, so the ordering is really done using (accept-stamp,server-ID) only), agent Q and M will have the same display order. (This display order may change later when anti-entropy actions are performed with other agents)

7. [10 points]: What other seat assignment displays are possible (in addition to the example in the previous question)?

Answer: The possible displays are:
1: Balakrishnan 2: Kaashoek 3: Morris (This ordering is possible if agent M's logical clock is ahead of agent Q's. i.e. the accept-stamp of Balakrishnan's reservation is lower than that of Kaashoek's. Balakrishnan's reservation could also have the same accept-stamp as Kaashoek's, but agent M has a lower server ID)
1: Kaashoek 2: Balakrishnan 3: Morris (This ordering is possible if Balakrishnan's reservation accept-stamp is in between of Kaashoek's or Morris'. Other explanations also exist)

8. [10 points]: Now Dr. Julius No approaches the agents and demands to be given seat 1. Agent Q says "My laptop won't let me do that." Dr. No won't take that for an answer, and exclaims that surely Agent Felix can get him seat 1. Who is right: Agent Q? Dr. No? Both? Neither? And why?

Answer: Both are correct.

A. **Agent Felix might get Dr. No seat 1. Since Felix has not performed anti-entropy with any other agent and hence does not know about other people's reservations, Dr. No will be the first reservation. Therefore, Dr. No will be assigned seat 1 locally by Felix. Felix can make this seat assignment permanent by rushing to the primary replica and synchronize with it (we assume no other agents have yet synchronized with primary replica yet)**
B. **Agent Q is also correct. Since agent Q has already assigned seat 1 to Kaashoek, Dr. No could not be given seat 1. (In fact, the lowest number seat Dr. No could get from agent Q is seat 4).**

IV TAOS

The point of the proof in Section 2.2 of "Authentication in the Taos Operating System" is to convince the file server FS that Bob approves of request RQ. This question explores some of the reasons for the complexity of the proof, which stem from its relationship to the cryptographic operations and message exchanges that the workstation and server must perform.

9. [10 points]: The proof could be simplified if equation (4) read K_{Bob} **says** $(C_{Bob} \Rightarrow K_{Bob})$; then FS could use this directly with (3) to deduce K_{Bob} **says** RQ. Does this modified approach require that Bob's private key is in workstation memory for the duration of Bob's login session? Answer yes or no, and briefly explain your answer.

Answer: Yes. "says" is equivalent to signing with private key. Therefore K_{Bob}^{-1} is needed to authenticate the statement that $(C_{Bob} \Rightarrow K_{Bob})$ whenever a new channel is established. Hence we need to keep K_{Bob}^{-1} in memory.

10. [10 points]: In theory, the channel C_{Bob} and steps (3) through (5) could be eliminated and replaced with the single message $(K_{ws}|K_{Bob})$ **says** RQ sent from the workstation to the server. Then the server could deduce (6) directly from this message and (2). Does Taos gain a performance benefit from using its more elaborate protocol? Answer yes or no, and briefly explain your answer.

Answer: Yes. The new method requires $K_{ws}|K_{Bob}$ to sign every RQ using public key cryptography which is an expensive operation. Old method requires only one signature when the channel is first setup. All messages on channel are encrypted and authenticated using less expensive cryptographic primitives

Name:

V Distributed Data Structures

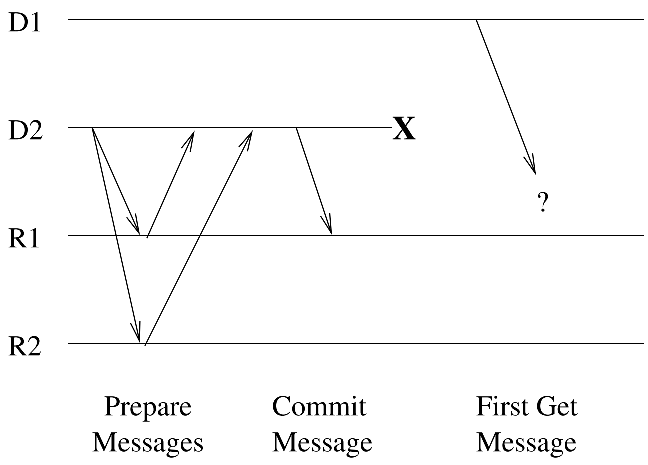
A distributed hash table, built using the system described in "Scalable, Distributed Data Structures for Internet Service Construction," has two replicas of the key/value pair corresponding to a particular key k_1. The value for k_1 is a boolean value, and starts out *false*. Imagine that a DDSlib, named D_1, issues a get for k_1, then pauses, then issues another get for k_1. Call the two results v_1 and v_2. Concurrently with with D_1, a different DDSlib, D_2, starts to execute put(k_1, *true*). No other gets or puts occur.

11. [10 points]: Section 2 of the paper mentions that DDS provides one-copy equivalence. One-copy equivalence turns out to allow three of the four combinations of two boolean values for (v_1, v_2). Which combination is forbidden, and why does one-copy equivalence forbid it? For this question you may assume that no failures occur.

(True, False) is forbidden in one-copy equivalence. If the first "get" operation of D_1 sees "True", this means the "put" operation has been executed. Hence all subsequent "get" operations will see the new value of "put" if there is only one copy of the data around

(Turn the page...)

12. [10 points]: D_2 crashes, after sending both prepare messages but only one commit message. You can assume no other nodes crash. D_1 executes its first get just after D_2 crashes. The figure below depicts the sequence of events. What sequences of results (v_1, v_2) are possible in the actual DDS implementation as described in the paper? For each sequence, briefly mention why it might occur.



Note that a replica can say "no" if it is not ready to give the answer. The possible results are:

- **A. (True,True): Both the first and second "get" request are sent to $R1$.**
- **B. (True, "no"): The first "get" request is sent to $R1$ and the second "get" request is sent to $R2$ before $R2$ times out and contacts $R1$.**
- **C. ("no","no"): Both the first and second "get" request are sent to $R2$ before $R2$ times out and contacts $R1$.**
- **D. ("no",True): The first "get" request is sent to $R2$ before $R2$ times out and contacts $R1$. The second "get" request is sent to $R1$**

Note that a node will lock its hash table (i.e. answers "no" to "get" request) after it receives a prepare message but before it gets the corresponding commit message.

Name:

13. [10 points]: Again, D_2 crashes after sending only one commit, and then D_1 executes its first get, but this time one of the replicas crashes just after D_1's first get completes. What sequences of results (v_1, v_2) are possible in this scenario? For each sequence, briefly mention why it might occur.

The possibilities are:

- **A. (True,True): The first "get" request is sent to $R1$ and sees True as the result. $R2$ crashes. The second "get" request is sent to $R1$ and sees True as the result.**
- **B. (True,False): The first "get" request is sent to $R1$ and sees True as the result. $R1$ crashes, $R2$ realizes the crash and continues..The second "get" request is sent to $R2$ and sees False as the result.**
- **C. ("no",False): The first "get" request is sent to $R2$ and gets the answer "no". $R1$ crashes, $R2$ realizes the crash and continues. The second "get" request is sent to $R2$ and sees False as the result.**
- **D. ("no",True): The first "get" request is sent to $R2$ and gets the answer "no". $R2$ crashes. The second "get" request is sent to $R1$ and gets True as the result.**
- **E. ("no","no"): The first "get" request is sent to $R2$ and gets the answer "no". $R1$ crashes but $R2$ has not timed out and realized the crash. The second "get" request is sent to $R2$ and gets "no".**

Comment: The fact that (True,False) is possible violates the one-copy equivalence claimed by the paper.

End of Final

Name: