Homework 9

due November 21, 2024

Problem 1:

Assume a middleware layer with reliable multicast delivery to nodes in a distributed system. Processes can still fail, they can appear to fail by being too busy to answer in time, or they can be restarted with the same process ID. Show how you can enhance any consensus protocol by making this arbitrary failure mode into stop-fail mode.

Background: The importance of Paxos is not so much in the protocol itself, but how Lamport showed how to make a simple protocol deal with more and more complicated failure modes.

Problem 2:

Assume a collection of processes that can propose values. A *consensus algorithm* ensures that

1. only one of the proposed values can be chosen.
2. Only a single value is chosen
3. A process never learns of a value chosen if this value has not been actually chosen.

We make the following assumptions.

1. Agents operate at arbitrary speed, may fail by stopping, and may restart.
2. Messages can take arbitrarily long to be delivered, can be duplicated, and can be lost, but they are not corrupted.

As in Paxos, processes can take three roles: Proposers, Acceptors, and Learners. As a proposer, a process can

Problem 2a:
What is the status of failure tolerance for a single proposer and a single acceptors.

Problem 2b:
We need to have more than one acceptor. We need to require a majority among acceptors to accept a proposal. Impose a rule that prevents two different values of getting a majority of acceptors.

Problem 2c:
Assume that the single proposer proposes a value. Give a scenario where the proposer does not know that the value has been accepted.

Problem 3:
The Needham Schroeder Protocol for a Key Distribution Center works in the following way:

(1) Alice sends request to KDC, stating her ID and the service she wants to contact

Alice to KDC: N_1, Alice, Bob

Here, N_1 is a nonce, a random value not to be reused.

(2) KDC sends Alice a message encrypted with the key shared by her and the KDC K_A

a. The nonce as a session identifier
b. The service name
c. A ticket for the service, consisting of an invented key for Bob and Alice, K_{AB}
d. The ticket can only be read by Bob: $K_B\left(K_{AB}, \text{Alice}\right)$ and therefore encrypted with the key that Bob shares with the KDC, K_B

KDC to Alice: $K_A\left(N_1, \text{Bob}, K_{AB}, K_B\left(K_{AB}, \text{Alice}\right)\right)$

(3) Only Bob can read the ticket. Now Alice and Bob can show that they both know K_{AB}, which indirectly shows that Alice knows K_A and must be Alice (or the KDC) and that Bob knows K_B and therefore has to be Bob (or the KDC).

(4) Alice sends the ticket $K_B\left(K_{AB}, \text{Alice}\right)$ to Bob and starts authentication by sending $K_{AB}(N_2)$ with a second nonce.

Alice to Bob: $K_B\left(K_{AB}, \text{Alice}\right)$ $K_{AB}(N_2)$

(5) Bob decrypts the ticket $K_B\left(K_{AB}, \text{Alice}\right)$. Bob now authenticates by using the common key K_{AB} to show that Bob can indeed read the ticket.

Bob to Alice: $K_{AB}(N_2 - 1, N_3)$

with yet a third nonce N_3.

(6) Alice decrypts and now knows that Bob is indeed Bob (or anyone knowing K_{AB}). She now proves herself with

Alice to Bob: $K_{AB}(N_3 - 1)$.

(7) Bob deciphers and now knows Alice since she solved the puzzle.

In summary, we have:

1.	Alice to KDC:	N_1, Alice, Bob
2.	KDC to Alice:	$K_A\left(N_1, \text{Bob}, K_{AB}, K_B\left(K_{AB}, \text{Alice}\right)\right)$
3.	Alice to Bob:	$K_B\left(K_{AB}, \text{Alice}\right)$ $K_{AB}(N_2)$
4.	Bob to Alice:	$K_{AB}(N_2 - 1, N_3)$
5.	Alice to Bob:	$K_{AB}(N_3 - 1)$

What vulnerability arises if we change message 4 to $K_{AB}(N_2 - 1), K_{AB}(N_3)$?