Tutorial
Distributed Systems (IN2259)

SAMPLE SOLUTION: EXERCISES ON BLOCKCHAIN

EXERCISE 1 - Byzantine Generals

Consider the following Byzantine generals example, with 5 generals. The 2 colored generals are faulty, while the rest are non-faulty. The correct generals wish to unanimously decide on the same action (attack or retreat), while the traitors work together to prevent them from doing so. Each general can communicate to any other general with a fixed delay (i.e., synchronous network).

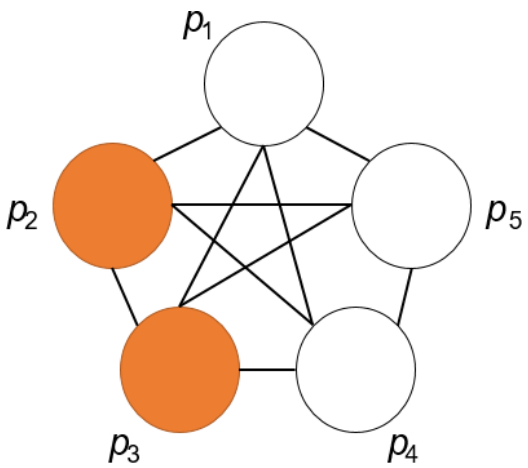

Figure 1.1: Byzantine generals example

(a) Assume $p1$ is the commander, while the others are lieutenants. Can Lamport's original solution to the problem work in this situation? If yes, explain why. If not, show a counter-example.

Solution: Lamport's solution works only for $3f+1$ nodes, where f is the number of faulty generals. In this situation, $f = 2$, there the minimum number of generals required is 7. Below is a sample execution which does not achieve consensus. The commander decides to attack and sends it all. p_4 and p_5 act normally and relay this information to the others. p_2 and p_3 send attack to p_4, which then decides to attack. p_2 and p_3 both send retreat to p_5. Since it receives 2 attacks, 2 retreats, it cannot decide between the two, and consensus is not achieved among the three non-faulty generals.

(b) Assume there are no commander. Describe an algorithm which achieves consensus in the above example with high probability. What happens if the number of traitors is increased to 3? Assume each general has equal computational power.

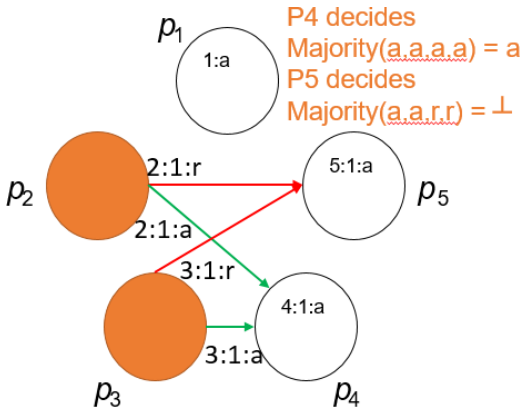

Figure 1.2: Counter-example to Lamport's solution

Solution:

```
initially do {
        Votes := NULL
        Prefer := proposed(i)
        r' := MaxNumberOfRounds
        r := 0
        compute (r)
}

on receive (Votes', Prefer') do {
        if verify (Votes', Prefer') and |Votes'| > |Votes| then {
                Votes := Votes'
                Prefer := Prefer'
                r := |Votes'|
                compute (r) // stop current computation and start the next round
        } else {
                continue current compute (r)
        }
}

on compute (r) do {
        if r >= r' then {
                decide Prefer
        }

        repeat until verify (Votes U {nonce}, Prefer) or stopped by a message {
                nonce := random(0,1) // try a random number
        }
        broadcast (Votes U {nonce}, Prefer)
        r++
        compute (r)
}
```

Listing 1.1: Blockchain consensus at process p_i

The function $verify(Votes, Prefer)$ is repeatedly called to assess if the current chain of $Votes$ solve the puzzle set by

the preferred value $Prefer.\ verify$ will check the entire list of votes, meaning each prefix of elements correctly solves the puzzle defined by the $Prefer$ value. The parameter r' is set to limit the number of computation rounds required to finish the computation. The probability of achieving consensus increases as the value of r' increases, at the cost of added computation effort. Initially, each general p_i starts with a preferred value $proposed(i)$.

In order to break consensus, the faulty generals must create one valid blockchain of length r' with a different $Prefer$ value from the blockchain worked on by the loyal generals and send their blockchain to one loyal general. That loyal general could then verify the validity of the traitors' blockchain and decide on a different value from the rest of the loyal generals, thus violating agreement. However, since there are more loyal generals than traitors, it is unlikely that they are able to produce one verifiable blockchain before the loyal ones are able to finish theirs.

If they control more than the majority (3 faulty generals), their computation advantage can be used to outpace the loyal generals and eventually produce the blockchain needed to break agreement, assuming r' is sufficiently high.

EXERCISE 2 - Merkle Tree

Consider the function $f(x) = (2 \times x) mod 100$. A Merkle Tree can be built using $f(x)$ as the hashing function, and the pairing of results is done by summation of the results (e.g., $f(f(x)+f(y))$).

(a) Draw the Merkle Tree for transactions 66, 20, 45, 59, 38, 2, 6, 100. The input of $f(x)$ is only the id of each transaction. The tree is sorted in ascending order.

Solution: Transactions are applied as is to $f(x)$ to form the leaf nodes. The leaf nodes are summed before applying to $f(x)$. For example, $f(f(2)+f(6)) = f(4+12) = f(16) = 32$. The complete solution is:

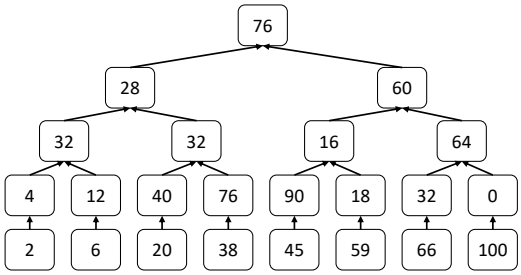

Figure 2.3: Merkle Tree solution

(b) Provide the Merkle proof for transaction 45. Then, show how transaction 45 is verified using the Merkle proof as in the Simple Payment Verification (SPV).

Solution: The Merkle proof contains the highlighted nodes:

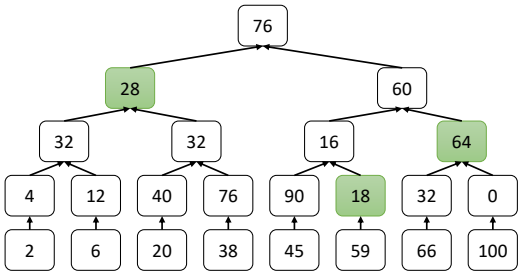

Figure 2.4: Merkle Proof

To verify 45, the SPV client hashes the transaction to find 90. Then it pairs it with 18 to find 16. It then pairs 16 with 64 to find 60. Finally, it pairs 60 with 28 to find 76. Since the SPV client knows the Merkle root of every block, it can determine that transaction 45 is included in this Merkle tree by verifying with the known value 76.

(c) Suppose that all the unspent transaction outputs (UTXOs) of transactions 20, 38, 6, and 2 are consumed in that order. Show how the Merkle tree can be safely pruned if it only tracks the transactions with UTXOs.

Solution: After transaction 20 is spent, 76 is removed. After 38 is spent, 40 and the leftmost 32 can be removed since they are no longer needed in any further Merkle proofs for unspent transactions. After 6 is spent, 4 is removed. Finally, after 2 is spent, 12, and the second 32 can be removed because only 28 is necessary for Merkle proofs of transactions on the right side of the tree. Furthermore, the hash value 60 from the right side is also removed. The final tree after the four transactions are removed is:

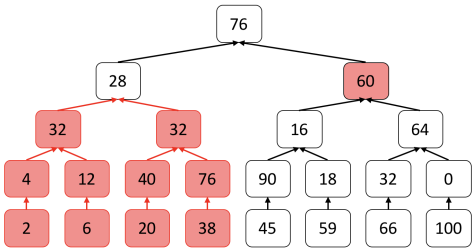

Figure 2.5: Merkle Tree pruned

EXERCISE 3 - Attacking Bitcoin

Consider "feather-forking", where a Bitcoin mining pool with a proportion $\alpha \in [0,1]$ of the global hash-rate blacklists a specific transaction by refusing to mine a new block on top of any block containing this transaction, and instead branches off with a different block that does not contain it. In a two-block feather-fork attack, the attacker abandons the attack once two confirmations are made. The following figure illustrates the attack:

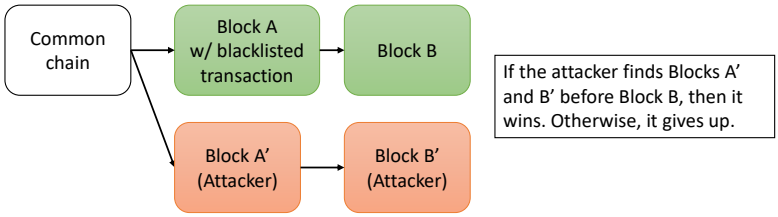

Figure 3.6: Feather-forking attack.

(a) Given $\alpha = 0.25$, what is the probability for a given feather forking attack to succeed? Assume that the probability of mining the next block is proportional to the hash-rate of the miner.

Solution: After block A is mined, the attacker is trying to block A'. The probability to mine A' as the next block is $\alpha = 0.25$. If A' is successfully mined, the attacker must then mine B'. According to our assumption, the probability to mine B' is then also $\alpha = 0.25$. Therefore, the overall probability of mining $A'B'$ as the next two blocks is $\alpha^2 = 0.25^2 = 0.0625 = 6.25\%$

(b) If honest miners are aware of a feather-forking attack, how should their mining decision be affected?

Solution: Honest miners collect a reward for each block successfully mined which stays in the blockchain permanently. However, if they mine a block which contains a transaction which is blacklisted by a feather forking attack, they risk losing the block reward if the attack is successful (with probability α^2). Honest miners therefore must decide whether to include a blacklisted transaction or not depending on its attached fee. If the fee is high enough, the miner is incentivized to include the transaction despite the risk of feather forking attack.

(c) Given a block reward of 12.5 BTCs and an average transaction fee of 0.1 BTCs, what should the blacklisted transaction fee be to subvert the attack? Assume each block contains 100 transactions and that $\alpha = 0.25$.

Solution: Honest miners can compute the following expected value for blocks, depending whether they contain the blacklisted transaction Tx with fee TxF or not:

$TotalReward = BlockReward + TransactionFees$

$TotalReward(NotIncluded) = BlockReward + 100 \times AverageTransactionFee = 12.5 + 100 \times 0.1 = 22.5$

$TotalReward(Included) = (1-\alpha^2) \times (BlockReward + 99 \times AverageTransactionFee + TxF) = (1-0.0625) \times (12.5 + 99 \times 0.1 + TxF) = 21 + 0.9375 \times TxF$

Honest miners will include Tx if $TotalReward(Included) \geq TotalReward(NotIncluded)$, then $21 + 0.9375 \times TxF \geq 22.5$. Therefore $TxF \geq 1.6$. The blacklisted transaction must cost around 16 times more than the average transaction!