Homework 4

Junhong Chen

Now suppose a blockchain using the Bitcoin protocol runs at a very fast block generation rate, so that only 40% of the generated blocks are on the main chain on average when the blockchain runs normally without attackers. Now there is an attacker controlling 33% of the total network hash rate. Is it possible for this attacker to deterministically launch double-spending attacks? Why?

The growth rate of the networks main chain (beta) equals the block generation rate (lambda) times 40%, so 40% is the security threshold of the system. The attacker is controlling 33 % of the total hash power so p =33%. When the p is less than the security threshold, the possibility of the attack launch the attack is small, but there is still possible for this attacker to deterministically launch double spending attacks if the attacker is only 1 or 2 blocks behind.

For the previous question, if the chain uses GHOST protocol instead. All the rest conditions stay the same. Is it possible for the attacker to deterministically launch double-spending attacks? Why?

It is less possible for the attacker to deterministically launch double spending attacks if using the GHOST protocol. Unlike the Bitcoin protocol, the GHOST protocol uses a different approach to select the main chain. It selects a child block with the largest subtree and then selects the path that has the most cumulative work done on it, which is resilient to the 50% attacks.

One claims that by using GHOST protocol, we can now simply lower the difficulty to speed up the block generation to achieve arbitrarily high transaction throughput, because fast block generation no longer harms the security. Is this claim true or not? Why?

No. In GHOST protocol, blockchain needs to wait for the fork to collapse to determine the main chain. If the block generation rate is very fast, then the collapse time may become very long, which will delay the transaction

confirmation time, leading to congestion, increased network latency, and destabilization of the network.

Why in GHAST protocol, sometimes we assign same weights to all blocks, but sometimes we assign different weights to different blocks?

Assigning the same weight to all makes it easier to calculate the heaviest subtree without having to account for varying weights. This approach is beneficial for maintaining a straightforward and transparent consensus mechanism. Assigning different weights to blocks can enhance the security of the blockchain. For example, blocks that are more difficult to produce (e.g., requiring more computational work) can be given more weight, making it harder for an attacker to manipulate the blockchain.

In Conflux, is it possible for an attacker to create a malicious block choosing a very early block as its parent block to disrupt the transaction total order? Why?

No. Conflux derives the total order of all blocks based on their epochs and their topological order. Although the attacker chooses a very early block as the parent block, the new block will still be in the new epoch.

How much hash power an attacker must control for him/her to deterministically launch double spending attacks in Conflux? Why?

It's hard to determine. When the attacker tries to launch a double spending attack, the attacker needs to revert a transaction in an epoch, which requires the attacker to revert the pivot chain block associated with the epoch. It also needs to compete with all honest nodes to revert an the old pivot chain block that is on the common pivot chain.

Suppose we run PBFT algorithm on a cluster of 21 machines. What is the maximum number of machines we will be able to tolerant for failure simultaneously?

The maximum number of machines we will be able to tolerant for failure simultaneously is the floor of (21-1)/3 , which is 6.

A node in PBFT waits for prepare/commit messages from two thirds of nodes (replicas) during the prepare/commit stages. What would happen if the node instead waits for prepare and commit messages from only a simple majority of other nodes? Would the modified PBFT be secure? If not, please explain with a counter example.

If a node were to wait for prepare and commit messages from only a simple majority of other nodes, the system would no longer be secure. A simple majority means that the system could only tolerate f faulty nodes out of f+1 total nodes, reducing the number of faulty nodes the system can handle. This decrease in fault tolerance makes the network more vulnerable to Byzantine faults. Imagine a system with 5 nodes where 2 are faulty. Under the original PBFT requirement (2/3 majority), at least 4 nodes must agree for a decision to be committed. However, if we switch to a simple majority requirement, only 3 out of 5 nodes would need to agree. In this setup, the 2 faulty nodes could potentially align with 1 correct node to form a majority. This majority could then commit an incorrect or malicious decision.

PBFT relies on the primary node to send out pre-prepare messages to drive the consensus process. What would happen if the primary node is malicious or fails? How does PBFT handle this situation?

The other node needs to detect whether the value sent by the primary node is correct. The view change protocol gets triggered when a sufficient number of nodes suspect that the primary is faulty, either due to malicious actions or a failure, which let the other node takes over as the primary. The selection process ensures that every node agrees on who the new primary should be, preventing split decisions.

Could PBFT skip the commit messages? Suppose nodes in PBFT commit-local directly after receiving 2f + 1 prepare messages. What could go wrong? Please explain with a counter example (Consider the case where a view change happens immediately after one or two nodes commit-locally for a request).

No. Skipping the commit phase in PBFT (Practical Byzantine Fault Tolerance) and moving directly to commit-local after receiving 2f+1 prepare messages could introduce issues related to consistency and agreement across the network, especially in scenarios involving view changes. This can result in a state inconsistency across the network, where some nodes have executed a request that others have not. For example, assume there are 4 nodes, and f is 1. 3 nodes will move directly to commit-local after receiving 2f + 1 prepare messages without waiting for the fourth node commit the request locally. After the view change, the fourth node happens to be the new primary , it might propose a sequence of requests that contradicts the state of the other nodes, leading to inconsistencies and potential failure of the consensus process.