ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Distributed Computing

HS 2015 Prof. R. Wattenhofer

# Distributed Systems Part II

**Solution to Exercise Sheet 6**

## 1 Is delayed Bitcoin strongly consistent?

**a)** It is true that naturally occurring forks of length $l$ decrease exponentially with $l$, however this covers naturally occuring blockchain forks only. It is possible for an attacker with a majority of the computational resources to go back to any point in time and compute an alternative blockchain. Eventually the attacker will overtake the currently active blockchain, thus replacing all changes since the block the attacker based its fork on. Delayed Bitcoin is therefore not strongly consistent, even if we were to transactions to be reverted/replaced with a negligible probability. In fact in 2014 one mining pool, Ghash.io, managed for a short period to maintain a share of computational resources larger than 50%, potentially subverting the blockchain, but not long enough to do any real damage.

**b)** The delay in this case prevents coins from completely vanishing in the case of a fork. Newly mined coins only exist in the fork containing the block that created them. In case of a blockchain fork the coins would disappear and transactions spending them would become invalid as well. It would therefore be possible to taint any number of transactions that are valid in one fork and not valid in another. Waiting for maturation ensures that it is very improbable that the coins will later disappear.

## 2 Doublespending

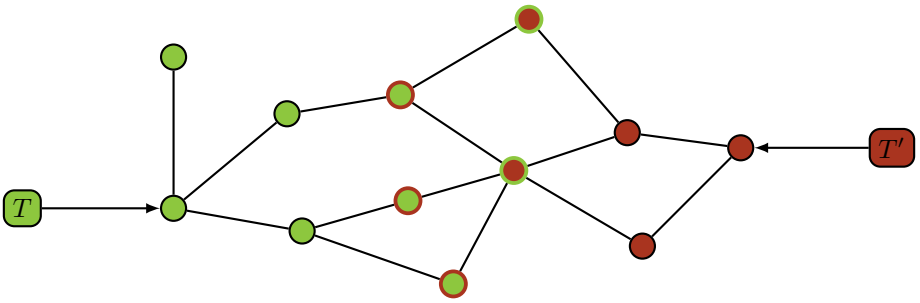


Figure 1: Random Bitcoin network

**a)** Figure 1 depicts the final situation. 7 nodes have seen $T$ first and 5 nodes have seen $T'$ first. The 5 nodes at the edge cut between the green and the red cut have seen both transactions.

**b)** Each node has 1/12 of all computational resources, hence the probability of $T$ being confirmed is $7/12 \approx 58\%$, while $T'$ has a $5/12 \approx 42\%$ chance of being confirmed. The higher connectivity from the first node seeing $T$ resulted in the transaction spreading faster, increasing the probability of winning the doublespend.

**c)** The first node that sees $T'$ now has 20% of the computational resources. $T'$ therefore has a probability to win of $2/10 + 1/11 \cdot 8/10 \cdot 4 \approx 49\%$. The distribution of computational resources in the network therefore matters. The goal of an attacker is to spread the transaction that she wants to have confirmed to a majority of the computational resources, which may not be the same as spreading it to a majority of nodes.

# 3 Partially spending outputs

Fully spending an output simplifies the bookkeeping considerably as an output can only be in two possible states: spent or unspent. This means that it is easy to detect conflicts, because two transactions spending the same output are conflicts. If we were to partially spend outputs, allowing multiple transactions to spend the same output until the coins on that output were completely spent, then the conflicts become more complicated. Assume an output with value 1 bitcoin. When partially spending outputs we could create 3 transactions claiming 0.5 bitcoins from that output, two of them are valid and the third will be invalid, but there are 3 possible combinations that are valid. So the simple answer is: it makes conflicts evident and reduces combinations for conflicts.

The number of possible combinations increases rapidly with the number of transactions and it becomes easy to build partition the network into very small partitions. An attacker could observe in which partition its victim it can construct followup transaction so that the transaction destined for the victim is valid only in the victims partition and nowhere else. This allows the attacker to maximize the possibility of its doublespend to be successful by limiting the spread of the victim's transacton and maximizing the spread of the attacker's transaction.

# 4 Replacement using sequence numbers

**a)** The original implementation of sequence numbers would broadcast a transaction before its timelock expires, and nodes would accept them into their mempool, but not mine them. The transaction could then be replaced by broadcasting a new version with a higher sequence number. This implementation had the problem that it is trivial for an attacker to perform a denial of service attack, by creating a rapid succession of transaction updates, which would then e forwarded, amplifying the attack.

**b)** Since the sequence number is attached to the input we may end up in a situation in which two transactions spend the same inputs, but one has a higher sequence number on the first input and the other has a higher sequence number on the second input. In this case it is not clear which transaction has precendence.