Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Distributed Computing

Systems@ETH Zürich

Autumn Term 2020

Prof. T. Roscoe, Prof. R. Wattenhofer

Computer Systems

Assignment 10

Assigned on: **November 27, 2020**

1 Quorum Systems

Quiz

1.1 The Resilience of a Quorum System

a) Does a quorum system exist, which can tolerate that all nodes of a specific quorum fail? Give an example or prove its nonexistence.

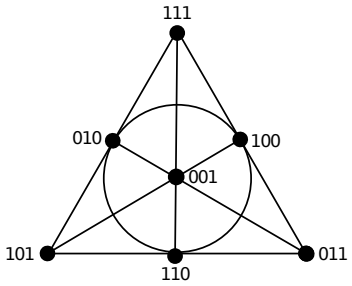
b) Consider the *nearly all* quorum system, which is made up of n different quorums, each containing $n-1$ servers. What is the resilience of this quorum system?

c) Can you think of a quorum system that contains as many quorums as possible?
Note: the quorum system does not have to be minimal.

Basic

1.2 A Quorum System

Consider a quorum system with 7 nodes numbered from 001 to 111, in which each three nodes fulfilling $x \oplus y = z$ constitute a quorum. In the following picture this quorum system is represented: All nodes on a line (such as 111, 010, 101) and the nodes on the circle (010, 100, 110) form a quorum.



a) Of how many different quorums does this system consist and what are its work and its load?

b) Calculate its resilience f. Give an example where this quorum system does not work anymore with $f+1$ faulty nodes.

Advanced

1.3 Uniform Quorum Systems

Definitions:

s-Uniform: A quorum system $\mathcal{S}$ is *s-uniform* if every quorum in $\mathcal{S}$ has exactly s elements.
Balanced access strategy: An access strategy Z for a quorum system $\mathcal{S}$ is *balanced* if it satisfies $L_Z(v_i) = L$ for all $v_i \in V$ for some value L.

Claim: An s-uniform quorum system $\mathcal{S}$ reaches an optimal load with a balanced access strategy, if such a strategy exists.

a) Describe in your own words why this claim is true.

b) Prove the optimality of a balanced access strategy on an s-uniform quorum system.

2 Eventual Consistency & Bitcoin

Quiz

2.1 Delayed Bitcoin

In the lecture we have seen that Bitcoin only has eventual consistency guarantees. The state of nodes may temporarily diverge as they accept different transactions and consistency will be re-estalished eventually by blocks confirming transactions. If, however, we consider a delayed state, i.e., the state as it was a given number Δ of blocks ago, then we can say that all nodes are consistent with high probability.

a) Can we say that the Δ-delayed state is strongly consistent for sufficiently large Δ?

b) Reward transactions make use of the increased consistency by allowing reward outputs to be spent after *maturing* for 100 blocks. What are the advantages of this maturation period?

Basic

2.2 Double Spending

Figure 1 represents the topology of a small Bitcoin network. Further assume that the two transactions T and T' of a doublespend are released simultaneously at the two nodes in the network and that forwarding is synchronous, i.e., after t rounds a transaction was forwarded t hops.

a) Once the transactions have fully propagated, which nodes know about which transactions?

b) Assuming that all nodes have the same computational power, i.e., same chances of finding a block, what is the probability that T will be confirmed?

c) Assuming the rightmost node, which sees T' first, has 20% of the computational power and all nodes have equal parts of the remaining 80%, what is the probability that T' will be confirmed?

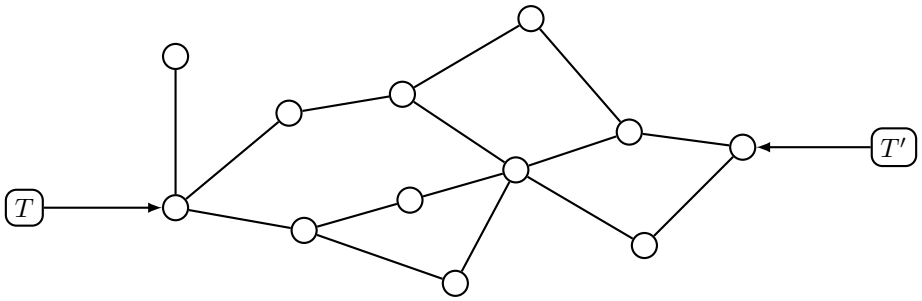


Figure 1: Random Bitcoin network

2.3 The Transaction Graph

In Bitcoin existing money is stored as 'outputs'. An output is essentially a tuple (address, value). A transaction has a list of inputs, which reference existing outputs to destroy and a list of new outputs to create.

Because of this construction inputs claim the entire value associated with an output, even if the intended transfer is for a much smaller value than what the input references. If the input claims a larger value than needed for the transfer the user simply adds a *change output*, which returns the excess bitcoins to an address owned by the sender.

a) Draw the transaction graph created by the following transactions. Assume no fees are paid to the miners. Draw transactions as rectangles and outputs as circles. Arrows should point from outputs to the transactions spending them and from transactions to the outputs they are creating.

(a) Address A mines 50 BTC.

(b) Address B mines 50 BTC.

(c) A sends 20 BTC to C.

(d) B sends 30 BTC to C.

(e) C sends 40 BTC to A.

b) Mark the still unspent transaction outputs (UTXO) in your graph.

c) Why do inputs always spend the entire output value and not just the part that is needed for the transfer? Assume you can spend parts of an output and explain what would be needed to validate transactions and prevent the illegal generation of money.

Advanced

2.4 Bitcoin Script

Bitcoin implements a simple scripting language called "Bitcoin Script" to give additional conditions on transactions apart from correct signatures. The scripts are evaluated by the miners, which reject transactions and blocks containing such scripts if the script evaluates with an error.

With scripts, a timelocked transaction could be created that states something like:

> A and B sign that they want to spend the existing outputs ref_1, ref_2, ref_3 and create new_output_1, new_output_2. This transaction is invalid in a block with a blockheight lower than 450,000.

Similarly it is possible to create outputs with more complex spending conditions. E.g., instead of requiring a valid signature to spend an output it is possible to require multiple signatures by different parties:

A and B sign that they want to spend the existing outputs ref_1, ref_2, ref_3 and create a new output that can be spent with two signatures corresponding to two of the three public keys pub_1, pub_2, pub_3.

Using timelocks and "multisig outputs" it is possible to replace uncommitted transactions by creating a first transaction with a large timelock and spending the same coins in a replacement transaction with a smaller timelock. The smaller timelock ensures that the second transaction can be executed before the first one becomes valid.

Building on this idea, a "payment channel"[1] can be created where money can be exchanged with someone else securely without doing every transaction on the blockchain. This works by replacing the transaction and moving money between the outputs. The construction is shown in Figure 2.

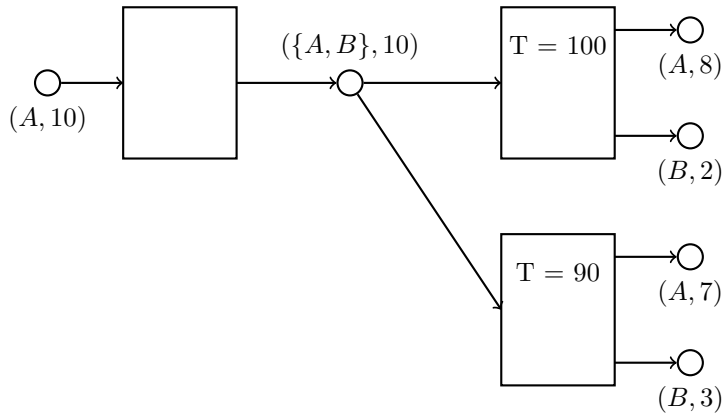


Figure 2: A "payment channel". A and B both have to sign to spend the output in the middle. The upper transaction can only be committed starting from blockheight 100, the lower one starting from blockheight 90.

Here first the left transaction is executed, this is called "opening" the channel. Then the transaction on the top right is prepared, but not executed. It can then be replaced by using lower timelocks.

a) What advantages does a payment channel have over regular Bitcoin transactions?

b) Why is the opening transaction needed? What could A do if the output being spent by the timelocked transactions would not require B's signature?

c) The channel cannot be used longer than the timeouts of the locktimes are. As soon as the first lock times out, the transaction needs to be executed, otherwise older replaced versions might become active as well. If someone wants to create a channel that he only uses occasionally, he needs to set the initial timelock far into the future.

Bitcoin also allows to define timelocks relative to the time the spent outputs were created. Can you think of a system that uses these relative timelocks to create channels that can be held open forever?

[1] Be careful, there are different versions with different features. In the lecture script are unidirectional channels. Here we are discussing an extended version: "Duplex Micropayment Channels". Duplex, because it is possible to move money in both directions.