Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Prof. T. Roscoe, Prof. R. Wattenhofer

Computer Systems

Assignment 12

1 Game Theory

Quiz

1.1 Selling a Franc

Form groups of two to three people. Every member of the group is a bidder in an auction for one (imaginary) franc. The franc is allocated to the highest bidder (for his/her last bid). Bids must be a multiple of CHF 0.05. This auction has a crux. Every bidder has to pay the amount of money he/she bid (last bid) – it does not matter if he/she gets the franc. Play the game!

a) Where did it all go wrong?

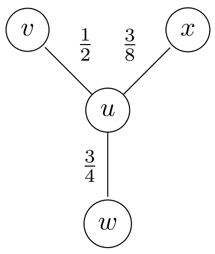
b) What could the bidders have done differently?

Basic

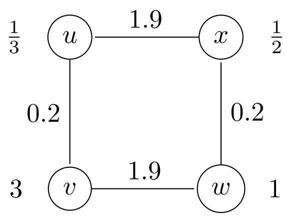
1.2 Selfish Caching

For each of the following caching networks, compute the social optimum, the pure Nash equilibria, the price of anarchy (PoA) as well as the optimistic price of anarchy($OPoA$):

i. $d_u = d_v = d_w = d_x = 1$



ii. The demand is written next to a node.



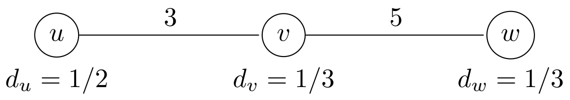
1.3 Selfish Caching with variable caching cost

The selfish caching model introduced in the lecture assumed that every peer incurs the same caching cost. However, this is a simplification of the reality. A peer with little storage space could experience a much higher caching cost than a peer who has terabytes of free disc space available. In this exercise, we omit the simplifying assumption and allow variable caching costs α_i for node i.

What are the Nash Equilibria in the following caching networks given that

i. $\alpha_u = 1$, $\alpha_v = 2$, $\alpha_w = 2$,

ii. $\alpha_u = 3$, $\alpha_v = 3/2$, $\alpha_w = 3$?



Does any of the above instances have a dominant strategy profile? What is the PoA of each instance?

Advanced

1.4 Matching Pennies

Tobias and Stephan like to gamble, and came up with the following game: Each of them secretly turns a penny to heads or tails. Then they reveal their choices simultaneously. If the pennies match Tobias gets both pennies, otherwise Stephan gets them.

Write down this 2-player game as a bi-matrix, and compute its (mixed) Nash equilibria!

1.5 PoA Classes

The PoA of a class $\mathcal{C}$ is defined as the maximum PoA over all instances in $\mathcal{C}$. Let

- $\mathcal{A}^n_{[a,b]}$ be the class of caching networks with n peers, $a \leq \alpha_i \leq b$, $d_i = 1$, and each edge has weight 1,
- $\mathcal{W}^n_{[a,b]}$ be the class of networks with n peers, $a \leq d_i \leq b$, $\alpha_i = 1$, and each edge has weight 1.

Show that $PoA(\mathcal{A}^n_{[a,b]}) \leq \frac{b}{a} \cdot PoA(\mathcal{W}^n_{[\frac{1}{b},\frac{1}{a}]})$ for all $n > 0$.

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-established 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.

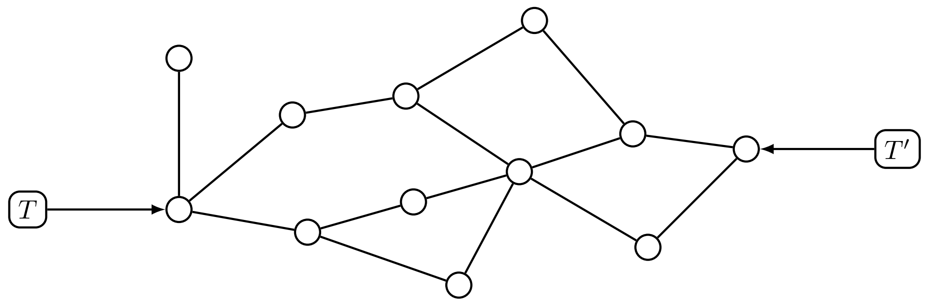


Figure 1: Random Bitcoin network

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., the 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?

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 block height 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.

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?

[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.

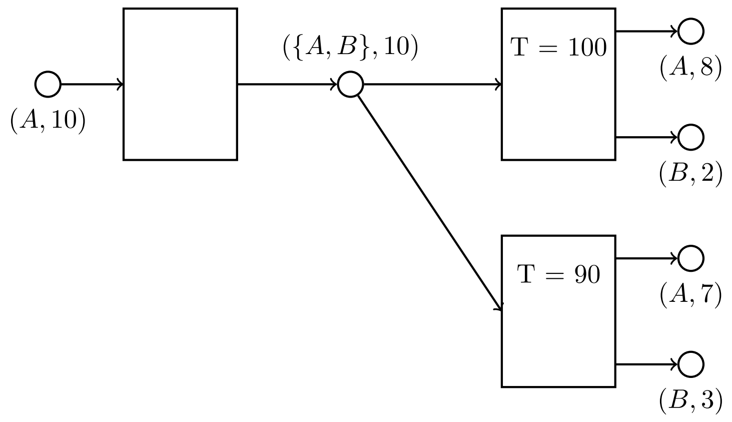


Figure 2: A “payment channel”. A and B both have to sign to spend the output in the middle. The upper transaction can only be committed starting from block height 100, the lower one starting from block height 90.

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 definitions of 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?

3 Advanced Blockchain

Advanced

3.1 Selfish mining

The analysis of selfish mining shows that the selfish miner receives block rewards disproportionate to his hashing power share. Argue why this could be profitable in practice.

Remark: This is intended as an open question that might not have a definite answer. It is intended to spark discussion and encourage a deeper understanding of the selfish mining problem.

3.2 Smart Contracts

Solidity is a high-level programming language that compiles down to the EVM (Ethereum Virtual Machine). Solidity's documentation[2] has an example cryptocurrency smart contract. Modify this smart contract so that the creator can add more minters and any of them can mint more coins. Deploy this modified smart contract on the Goerli Ethereum test network. A test network for Ethereum resembles the main network, but the Ether that is used in this network has no value. The test network is used for development/testing purposes. Through the smart contract ABI (Application Binary Interface), invoke the call to add another minter, and a follow-up call to let this new minter mint new coins for another unrelated address.

- Download MetaMask and create a Wallet[3]. This is a browser extension that allows you to interact with the Ethereum network.
- Get some test Ether from the Alchemy Goerli Faucet[4]. You need to create an Alchemy account to get the test Ether.
- Go to the Remix IDE[5] and create a new file. Copy the smart contract code from the Solidity documentation into this file and modify it.
- Compile the smart contract and deploy it on the Goerli test network.
- If no errors are thrown go to the "Deoploy & Run Transactions" section. Choose "Injected Provider - Metamask" as your environment and you will be prompted to connect the Metamask extension to the Remix IDE. Select an account and set your gas limit to 3000000. Click on "Deploy" and confirm the transaction in Metamask.
- Now your contract is live on the test network and you have the option to execute the functions that you coded for your contract.
- You can use etherscan[6] to monitor the transactions on the test network.

[2]`https://solidity.readthedocs.io/en/latest/introduction-to-smart-contracts.html#subcurrency-example`
[3]`https://metamask.io/`
[4]`https://goerlifaucet.com/`
[5]`https://remix.ethereum.org/`
[6]`https://goerli.etherscan.io/`