Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Distributed Computing

Systems@ETH Zürich

Prof. T. Roscoe, Prof. R. Wattenhofer

Computer Systems

— Solution to Assignment 12 —

1 Game Theory

Quiz

1.1 Selling a Franc

We assume that there are two bidders, b_1 and b_2. If b_1 bids 5 rappen, his gain is 95 rappen. Now b_2 is inclined to bid 10 rappen and gains 90 rappen. This can continue until b_1 bids 95 rappen. Bidder b_2 now has the choice of losing 90 rappen (her last bid) or coming out even. Since she is a rational player, she will bid 1 franc. Bidder b_1 now faces a similar choice. Either he loses 95 rappen or he bids and has a chance of only losing 5 rappen. Since he is a rational player, he will bid 1.05 franc. This bidding war will continue indefinitely (or until one bidder runs out of money).

There are a few ways the bidders could have avoided this situation. Apart from the obvious, simply do not play, they could have also colluded. One bidder bids 5 rappen for the franc and the bidders will simply split the money they made. This requires that the bidders can trust each other. As you can guess, there are games that anticipate collaboration.
There exists however a strategy, which is profitable even for a non-colluding bidder. If the first bidder bids 95 rappen, she will win 5 rappen, because nobody else will also bid. Why will nobody else bid? If another bidder bids more, he will certainly not bid more than 1 franc, because this will yield a negative payoff. Instead, he could bid 1 franc. But then, the first bidder will bid 1.05 francs to minimize her loss. And then the game continues as outlined at the beginning and both bidders will incur a loss. Therefore, no rational player will bid 1 franc in this scenario.

Basic

1.2 Selfish Caching

To be sure that we find every Nash Equilibrium, we explicitly write down every best response.

i. The best response strategies are

   u: cache only if nobody else does. (B1)
   v: cache if neither u nor x cache. (B2)
   w: cache unless u caches. (B3)
   x: cache if neither u nor v cache. (B4)

Nash equilibrium. If we assume that u plays $Y_u = 1$ (u caches) the system can only be in a NE if $Y_v = Y_w = Y_x = 0$ due to (B1). Since for all v, w, and x it is the best response not to cache if u does, $x = (1000)$ is an Nash equilibrium. If $Y_u = 0$ then (B3) implies $Y_w = 1$. If furthermore, $Y_v = 1$ it must hold that $Y_x = 0$ due to (B2). This does not conflict with (B4), and (0110) constitutes another NE. Last, if $Y_v = 0$ then (B2) implies $Y_x = 1$, which is also okay with (B4). Hence (0011) is also a NE.

$$NE = \{(1000), (0110), (0011)\}$$

Price of anarchy. The social optimum is achieved in strategy profile (1000), namely $OPT = cost(1000) = 1 + \frac{1}{2} + \frac{3}{8} + \frac{3}{4} = \frac{21}{8}$. Since (1000) is also a Nash equilibrium we immediately get that $OPoA = 1$. The worst-case price of anarchy is

$$PoA = \frac{cost(0110)}{OPT} = \frac{\frac{1}{2} + 1 + 1 + \frac{7}{8}}{\frac{21}{8}} = \frac{9}{7} \approx 1.286.$$

ii. The best response strategies are

u: cache only if nobody else does. (B1)

v: cache unless u caches. (B2)

w: cache unless x caches. (B3)

x: cache if neither u nor w cache. (B4)

Nash equilibrium. If we assume that u plays $Y_u = 1$ (u caches) the system can only be in a NE if $Y_v = Y_w = Y_x = 0$ due to (B1). However, $Y_x = 0$ implies that $Y_w = 1$ due to (B3), and hence there can be no NE with $Y_u = 1$. In any NE it must hold that $Y_u = 0$. Consequently, it must hold that $Y_v = 1$ from (B2). Now if $Y_w = 1$ (B3) implies that x does not cache. This does not infringe rule (B4), and thus $x = (0110)$ is a Nash equilibrium. If $Y_w = 0$ then (B4) implies that x caches. As thus, rule (B3) is not violated $x = (0101)$ is also a Nash equilibrium.

Price of anarchy. The social optimum is achieved in strategy profile (0110), namely $OPT = cost(0110) = \frac{1}{3} \cdot 0.2 + 1 + 1 + \frac{1}{2} \cdot 0.2 = 2.1\overline{6}$. Since (0110) is also a Nash equilibrium we get that the optimistic price of anarchy is 1. The worst-case price of anarchy is

$$PoA = \frac{cost(0101)}{OPT} = \frac{1/3 \cdot 0.2 + 1 + 0.2 + 1}{2.1\overline{6}} = \frac{68}{65} \approx 1.046$$

1.3 Selfish Caching with variable caching cost

We define D_i to be the set of nodes that cover node i. A node j *covers* node i if and only if $c_{i \leftarrow j} < \alpha_i$, i.e., node i prefers accessing the object at node j than caching it. Convince yourself that a strategy profile is a Nash Equilibrium if and only if for each node i it holds that

- if $Y_i = 1$ then $Y_j = 0$ for all $j \in D_i$, and
- if $Y_i = 0$ then $\exists j \in D_i$ with $Y_j = 1$.

i. $D_u = \emptyset$, $D_v = \{u, w\}$, $D_w = \{u\}$. D_u being empty implies $Y_u = 1$ (i.e. caches the file). Hence $Y_v = 0$, and $Y_w = 1$. $NE = \{(101)\}$. $PoA = 1$ since (101) is also the social optimum strategy.

ii. $D_u = \{v\}$, $D_v = \{u\}$, $D_w = \{u, v\}$. If $Y_u = 1$, then $Y_v = 0$ and $Y_w = 0$. If $Y_u = 0$, then $Y_v = 1$. Hence $Y_w = 0$. The equilibria are $NE = \{(100), (010)\}$.

$$PoA = \frac{cost(100)}{cost(010)} = \frac{3 + 1 + 8/3}{3/2 + 3/2 + 5/3} = \frac{40}{28} \approx 1.43$$

Dominant strategies. Every dominant strategy profile is also a Nash equilibrium. Hence we only have to check the computed NEs whether they consist of dominant strategies only.

Let us consider game i. Since every dominant strategy profile is also a Nash Equilibrium, it suffices to consider the NE. The game has no dominant strategy profile. Profile (101) is no dominant strategy profile in game i. since, although $Y_u = 1$ is the dominant strategy for u, $Y_v = 0$, and $Y_w = 1$ are not dominant strategies for v and w. If $Y_v = 1$, then it would be the best response of w to set $Y_w = 0$. Game ii: Since the decision of node u whether to cache depends on the decision of node v, this is not a dominant strategy. Therefore, this game has no dominant strategy profile.

Advanced

1.4 Matching Pennies

The bi-matrix of the game with Tobias as row player, and Stephan as column player looks as follows:

	H	T
H	1-1	-11
T	-11	1-1

This zero-sum game has no pure Nash equilibrium. For the mixed NEs, Tobias plays heads (H) with probability p, tails (T) with probability $1-p$. Stephan plays H with probability q, and T with probability $1-q$. We get the expected utility functions Γ:

$$\begin{aligned}
\Gamma_T(p,q) &= p(q-(1-q))+(1-p)(-q+(1-q)) = (4q-2)\cdot p+1-2q \\
\Gamma_S(p,q) &= q(-p+(1-p))+(1-q)(p-(1-p)) = (2-4p)\cdot q+2p-1
\end{aligned}$$

If Stephan plays $q = 1/2$ the term $4q-2$ equals 0, and any choice of p will yield the same payoff for Tobias. If Tobias plays $p = 1/2$ then any choice of q is a best response for Stephan. Thus $(p,q) = (1/2,1/2)$ is a mixed NE. Note that for any choice of $p > 1/2$, Stephan's best response is to choose $q = 0$. For a $p < 1/2$ Stephan would choose $q = 1$. However, Tobias' best response to $q > 1/2$ is $p = 1$, and $p = 0$ if $q < 1/2$. Hence $(p,q) = (1/2,1/2)$ is the only pair of mutual best responses.

1.5 PoA Classes

Let I^n be an instance of $\mathcal{A}^n_{[a,b]}$ that maximizes the price of anarchy, i.e. $PoA(\mathcal{A}^n_{[a,b]}) = PoA(I^n)$. Let $x, y \in X$ be two strategy profiles in I^n such that $PoA(I^n) = cost(y)/cost(x)$. We show the claim by constructing an instance $\hat{I}^n \in \mathcal{W}^n_{[\frac{1}{b},\frac{1}{a}]}$ out of I^n for which it holds that $PoA(\hat{I}^n) \geq \frac{a}{b}PoA(I^n) = \frac{a}{b}PoA(\mathcal{A}^n_{[a,b]})$. We construct $\hat{I}^n$ by setting $d_i = 1/\alpha_i$, $\hat{\alpha}_i = 1$ where α_i are the placement costs (for local caching) of player i in I^n. All edges remain as in I^n. This game has the same Nash equilibria as I^n since the cover sets D_i (nodes for which we do not cache if these cache already) for each peer stay the same. A peer j is in D_i iff $c_{i\leftarrow j} < \alpha_i$, or $c_{i\leftarrow j}/\alpha_i < 1$ respectively. We get the bound by comparing the performance of the two strategies x, y that produce the PoA in I^n in $\hat{I}^n$. Note that x is not necessarily a social optimum in $\hat{I}^n$, but y is a Nash equilibrium also in $\hat{I}^n$, because the cover sets are the same.

$$\begin{aligned}
PoA(\hat{I}^n) &\geq \frac{\hat{cost}(y)}{\hat{cost}(x)} = \frac{\sum_{i=1}^n \left(y_i + (1-y_i)\frac{c_i(y)}{\alpha_i}\right)}{\sum_{i=1}^n \left(x_i + (1-x_i)\frac{c_i(x)}{\alpha_i}\right)} && (1)\\
&= \frac{b \cdot a \sum_{i=1}^n \left(y_i + (1-y_i)\frac{c_i(y)}{\alpha_i}\right)}{b \cdot a \sum_{i=1}^n \left(x_i + (1-x_i)\frac{c_i(x)}{\alpha_i}\right)} && (2)\\
&\geq \frac{a \sum_{i=1}^n (y_i\alpha_i + (1-y_i)c_i(y))}{b \sum_{i=1}^n (x_i\alpha_i + (1-x_i)c_i(x))} && (3)\\
&= \frac{a \cdot cost(y)}{b \cdot cost(x)} = \frac{a}{b} PoA(I^n) && (4)
\end{aligned}$$

$\hat{cost}(x)$ denotes the cost function in $\hat{I}^n$. x_i, and y_i are either 1 or 0. x_i equals 1 if player i caches in strategy profile x, and 0 if she does not. With $c_i(y)$ we denote the cost of node i if it access the file remotely in strategy y. For step (3) we exploit the fact that $b \geq \alpha_i$ and $a \leq \alpha_i$ for all i.

2 Eventual Consistency & Bitcoin

2.1 Delayed Bitcoin

a) It is true that naturally occurring forks of length l decrease exponentially with l, however this covers naturally occuring blockchain forks only. As there is no information how much calculation power exists in total, it is always possible a large blockchain fork exists. This may be the result of a network partition or an attacker secretly running a large mining operation.

This is a general problem with all "open-membership" consensus systems, where the number of existing consensus nodes is unknown and new nodes may join at any time. As it is always possible a much larger unknown part of the network exists, it is impossible to have strong consistency.

In the Bitcoin world an attack where an attacker is secretly mining a second blockchain to later revert many blocks is called a 51% attack, because it was thought necessary to have a majority of the mining power to do so. However later research showed that by using other weaknesses in Bitcoin it is possible to do such attacks already with about a third of the mining power.

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 accidentially.

Note that this is however only a protection against someone accidentially sending you money that disappears with a discontinued fork. The same thing can still happen, if someone with evil intent double spends the same coins on the other side of the fork. You will not be able to replay a transaction of a discontinued fork on the new active chain if the old owner spent them in a different transaction in the meantime. To prevent theft by such an attacker you need to wait enough time to regard the chance of forks continuing to exist to be small enough. A common value used is about one hour after a transaction entered a block ($\sim$6 blocks).

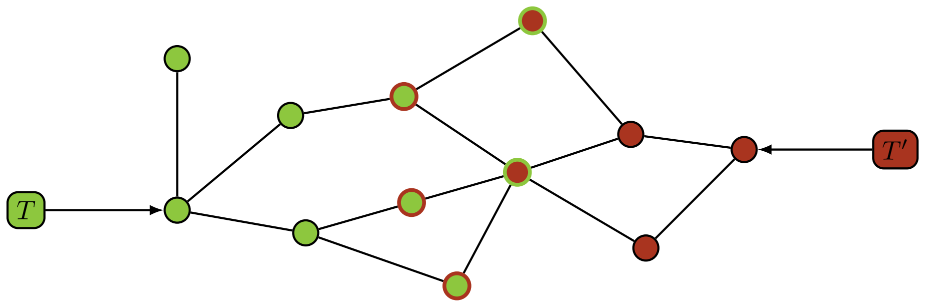

Figure 1: Random Bitcoin network

2.2 Double Spending

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.

2.3 The Transaction Graph

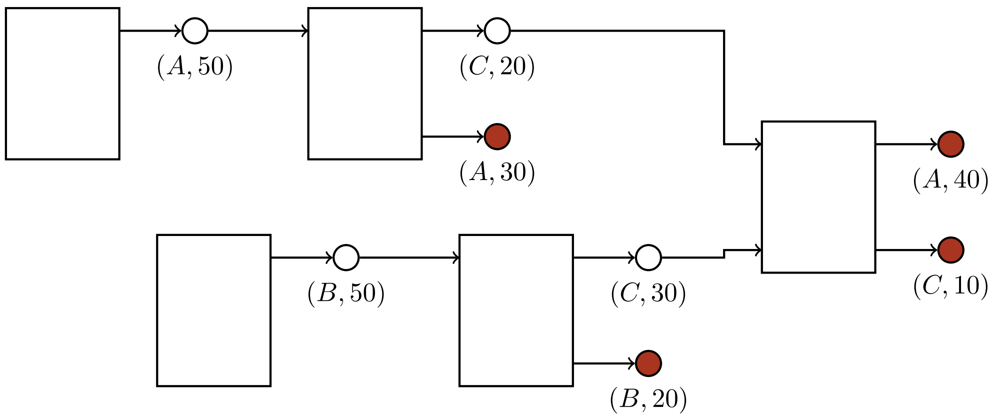

Figure 2: Transaction Graph. The red outputs are UTXOs.

a) See Figure 2.

b) See red outputs in Figure 2.

c) 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.

There are many points in the Bitcoin software where this would complicates things. A miner needs to construct a valid block from the known transactions, however this becomes more difficult if arbitrary combinations of transactions suddenly conflict with each other. Furthermore with complex conflicts attackers can create a situation where most nodes do not agree on the transactions which will be in the next block. However nodes are optimized to quickly be able to forward new blocks which look "expected". If the nodes do not agree at least loosely on the transactions to be committed in the next block, the propagation delays become much larger as many transactions need to be retransmitted, which finally results in smaller total transaction processing capability.

Note: As you might have realized the flow of money can be nicely followed with the transaction data from the Bitcoin blockchain. If some hacker steals your valuable coins, you can watch him buy things with it and see where the vendors are spending this money too! For this reason it is often possible to "buy" larger amounts of Bitcoin with less Bitcoin. The larger amount you can "buy" has very likely been involved in some crime and is being tracked by the police, thus the owner is eager to exchange them for other coins. Look for "Bitcoin doubling" in your favorite list of darknet services!

2.4 Bitcoin Script

a) Transactions are instantly finalized, so the large confirmation delay of the blockchain is irrelevant. Only the signatures of both parties are needed, then the money has effectively changed the owner. Furthermore no transaction fees have to be paid to miners for replacing a transaction.

b) Without the opening transaction A could just spend the money with a transaction without a timelock to a different address owned by himself. Requiring both signatures prevents this and gives security to B. In this construction B can trust that the funds will be available after the first timelock runs out.

Note that if B wants to access the funds earlier, it is still possible for A and B together to sign a transaction which directly executes the latest state. As long as both agree it is thus not necessary to wait for the timelocks. The timelocks are only necessary to ensure the last state in case there is disagreement.

c) A "kickoff" transaction can be introduced after the opening. Only the opening is executed (i.e., sent to the blockchain) at the beginning to secure the funds. Now transactions can be replaced and if someone wants to close the channel he can execute the kickoff. This starts the timers on the subsequent transactions. See Figure 3 for the new transactions.

In detail the protocol is the following.

Setup:

(a) A creates all transactions of the setup (opening, kickoff and first state).

(b) A sends these together with signatures for the kickoff and first state to B.

(c) B signs the kickoff and first state and sends the signatures to A.

(d) A signs the opening transaction and executes it on the blockchain.

Updating:

(a) A creates a new transaction spending the kickoff output with a lower timelock.

(b) A signs it and sends it to B.

(c) B sends his signature to A.

After the update both have a signed version of the new state and can terminate the channel in this state.

Closing:

(a) A or B proposes a settlement transaction that directly spends the locked-in funds in the opening output.

(b) The other party signs and sends it to the blockchain.

This is the cooperative closing case. If some dispute happens, either of the two parties can always send the kickoff and latest state to the blockchain.

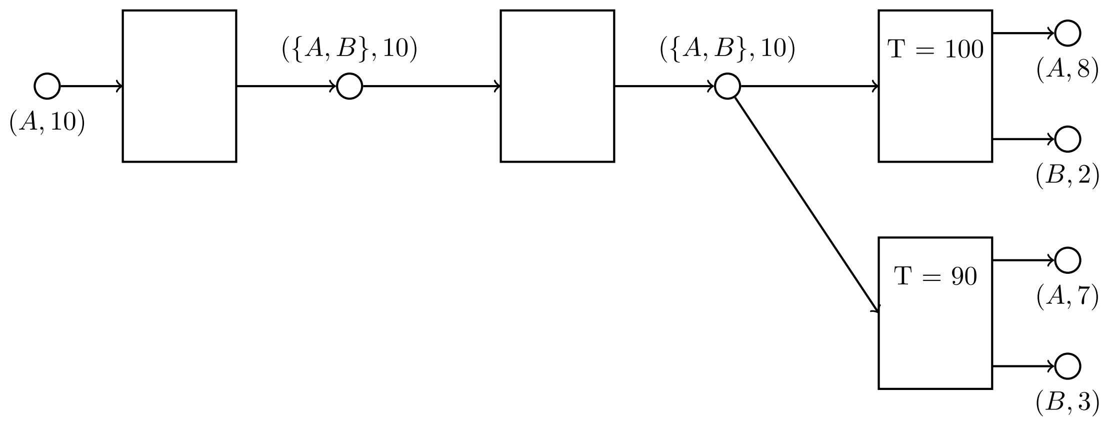

Figure 3: A "payment channel". A and B both have to sign to spend the output in the middle. The upper transaction can only be committed starting from blockheight 100, the lower one starting from blockheight 90.

3 Advanced Blockchain

3.1 Selfish mining

Here's our (somewhat subjective and non-conclusive) answer to this question:
Until the next difficulty adjustment, the block generation rate with respect to the longest chain is reduced in the presence of a selfish miner as the total hashing power would be distributed on two forks of the chain. Here, the selfish miner might even get unlucky and make less profit than by following the protocol.

However, due to the difficulty adjustment, the rewards distributed among the miners finding blocks will be reestablished to match a constant rate. From the Selfish Mining Theorem in the script we know that it is rational (profitable) for a miner to mine selfishly if it has enough hashing power (e.g. $\alpha \geq \frac{1}{3}$ or even some smaller α with $\gamma > 0$).

Further thoughts:

- Assume that miners reinvest some of their income in buying new mining equipment. As the global rewards per hour are kept at a constant level due to the difficulty adjustment and a selfish miner would get disproportionally high rewards, the honest miners must be awarded disproportionally small rewards. This would mean that the selfish miner will have a greater increase in his hashing power in comparison to the honest miners. Consequently, his share of block rewards will increase even more.
- Assume that other miners are rational (miners trying to maximize their own profit). They would be better off joining the selfish miner. This poses the potential for a majority attack.

Remark: There are papers that compare selfish mining rewards vs. honest mining rewards over a longer term taking Bitcoin's difficulty adjustment into account. For example: `https://arxiv.org/abs/1912.01798` *and* `https://eprint.iacr.org/2018/1084.pdf`.

3.2 Smart Contracts

In the exercise description we already suggested some tools to achieve the given task, but of course other methods can be used. With tools like Ganache [1] you can simulate a blockchain locally and test your smart contracts without deploying. Development environments like Hardhat [2] can further help you with testing and deployment of your code.
In the following we list the changes that you have to make to the original contract to achieve the wanted additional functionality:

- Add a new mapping to check if a wallet that sent the transaction is allowed to mint:

```
mapping (address => bool) public additional_minters;
```

- Changing the require statement on line 23 to:

```
require(msg.sender == minter || additional_minters[msg.sender] == true);
```

- Add a new function to add additional minters:

[1] https://trufflesuite.com/ganache/
[2] https://hardhat.org/

```
function add_additional_minter(address additional_minter) public {
     require(msg.sender == minter);
     additional_minters[additional_minter] = true;
}
```

The above tools show that the actual building/deploying of smart contracts on Ethereum is quite straightforward. Smart contracts themselves can be complicated and need to be extensively reviewed/tested/formally verified. Many third party libraries provide standard functionality that is tested extensively beforehand. Most of the times, it's recommended to use these libraries. But sometimes, it can be quite catastrophic - see `https://www.parity.io/a-postmortem-on-the-parity-multi-sig-library-self-destruct/`.