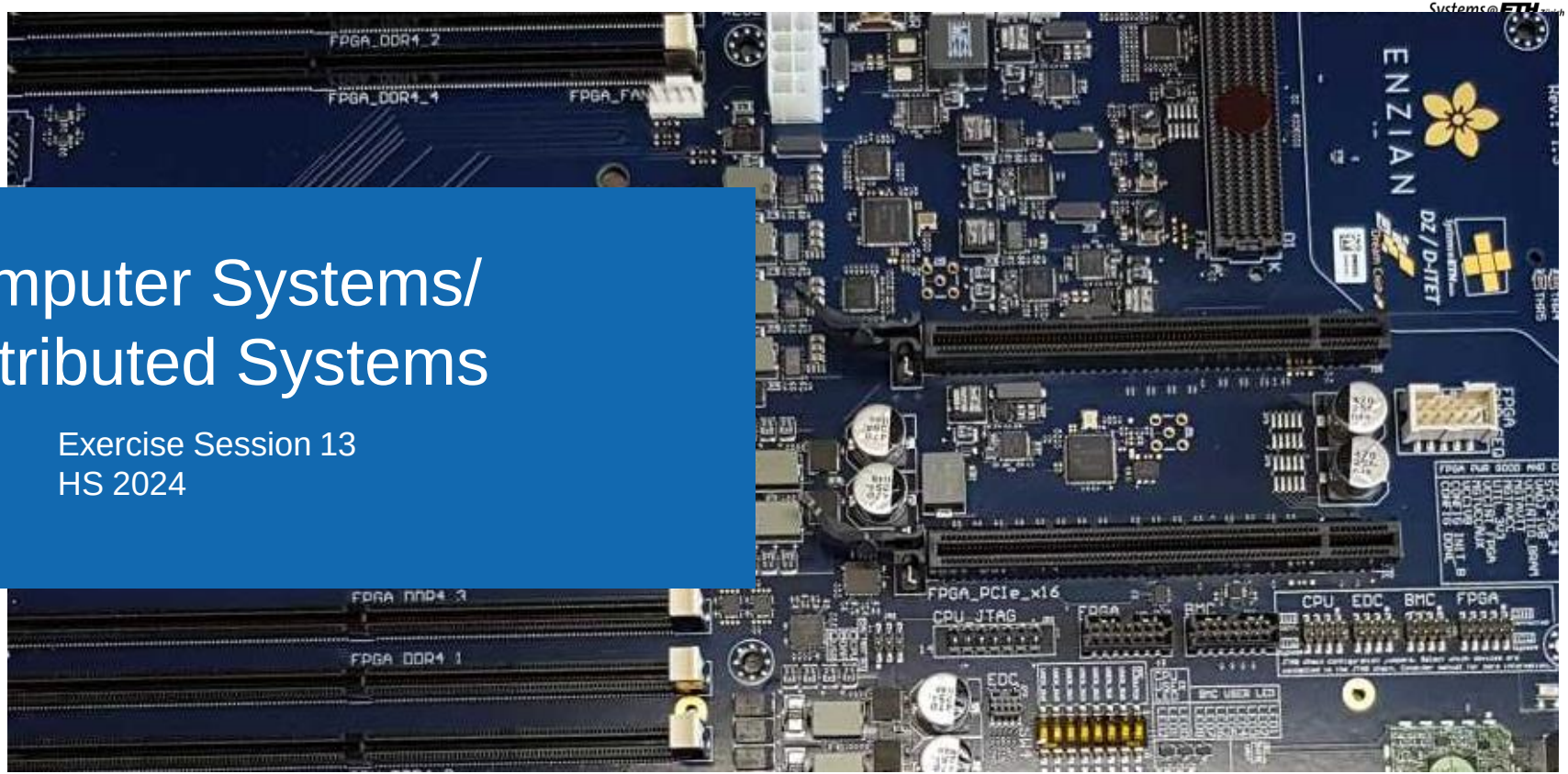
Slides last updated:
12.12.2024

Computer Systems/ Distributed Systems

Exercise Session 13
HS 2024



Program

1. Assignment review
2. Script recap
   - Game Theory
   - Eventual Consistency & Bitcoin
3. Quiz
4. Assignment preview

Assignment review

1. Clock Synchronization
    1. Topology
    2. Spanning Tree
2. Distributed Storage
    1. Hypercubic Networks
    2. Iterative vs. Recursive Lookup
    3. Building a set of Hash functions
    4. Multiple Skiplists

Assignment review – Clock Sync

Systems@ETH Zürich

Distributed Computing

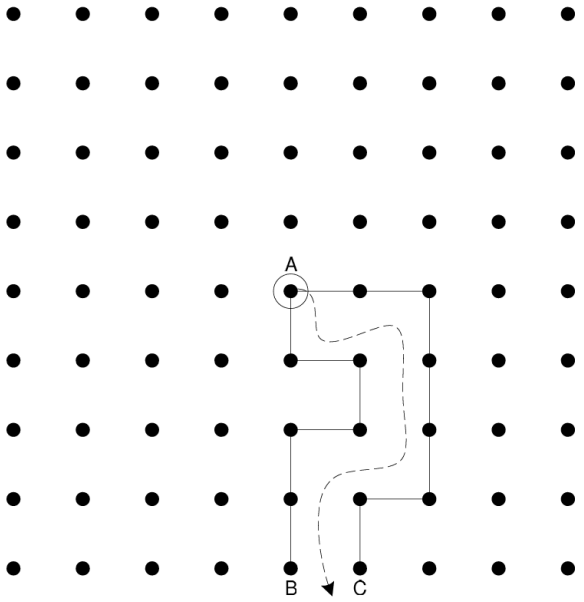
1.1)

Algorithm 23.9 Clock synchronization algorithm (code for node v)

```
1:  Increase clock C_v at local clock rate
2:  upon clock value C_v reaches next integer value:
3:      Send C_v to all neighboring nodes
4:  end upon
5:  upon receiving clock value C_w from node w:
6:      if C_w > C_v then
7:          C_v := C_w
8:          Send C_v to all neighboring nodes
9:      end if
10: end upon
```

Assignment review – Clock Sync

1.2)



Assignment review – Dist. Storage

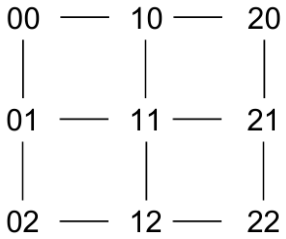
Systems@ETH Zürich

Distributed Computing

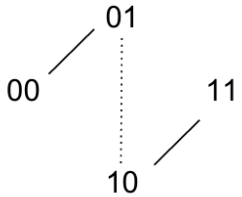
2.1)

M(3,1)

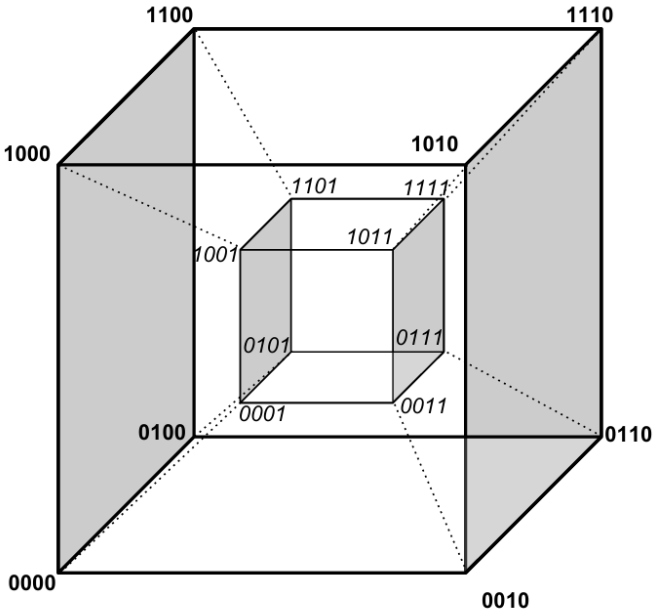
M(3,2)



SE(2)



M(2,4)



Assignment review – Dist. Storage

Systems@ETH Zürich

Distributed Computing

2.2)

a) In the recursive lookup there is no difference between a request originating at the node and a request being forwarded. Only once the lookup is finished do we need to care about forwarding the result to the previous node or returning it to the caller. This allows the same lookup logic to be reused for both. Furthermore, if the response is returned through the same path as the request was sent through, then the intermediate nodes can cache the result, potentially speeding up future lookups and distributing the load of a popular item.

b) Recursive lookups can easily be misused to mount Denial-of-Service attacks, since a single request message from an originating node is forwarded over multiple hops. Each hop multiplies the impact this message has on the network. Thus the attacker's bandwidth is potentially multiplied by the number of hops the request is routed through. Furthermore, if the result is returned over the same path as the request, then the attacker is hidden behind a number of hops and the victim only sees traffic originating from the last hop.

Assignment review – Dist. Storage

Systems@ETH Zürich

Distributed Computing

2.3)

The salted hashing function derivation allows random access to any of the derived hashing functions without having to recompute the intermediate functions, as is the case in the iterative hashing function derivation scheme. Furthermore, iterative hashing allows a user which knows the first hash value to deduce all the others by repeatedly applying h. This information can be used by an attacker to target specific nodes storing a movie.

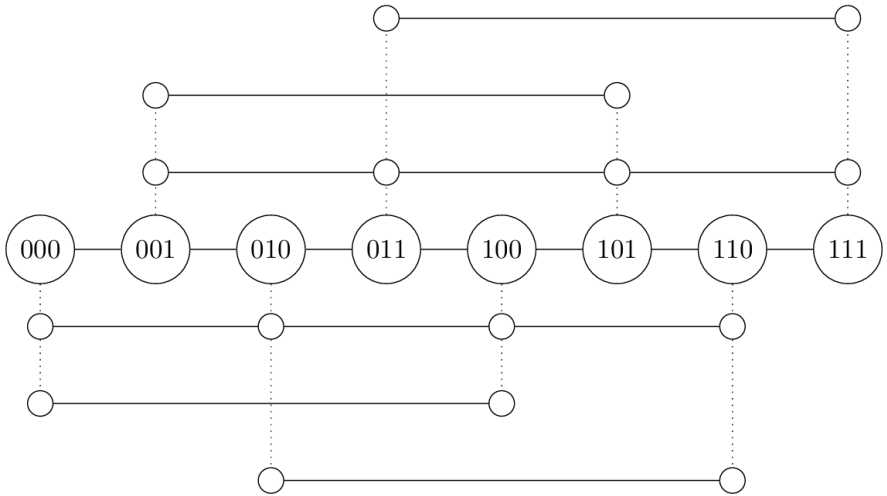
Assignment review – Dist. Storage

2.4)

a) Figure 3 shows the structure of a multi-skiplist with 8 nodes and 3 levels. Notice that each of the lists would wrap around at the ends.

b) Unlike in the single skiplist each node now has a constant degree of $2 \cdot d$, i.e., on each level it has a right and a left neighbor, including the full list at $l = 0$.

c) The number of hops is still $O(\log n)$, just like for the simple skiplist.



Game Theory

Nodes no longer have a common goal – They act selfish

- Prisoner's Dilemma
- Social optimum, dominant strategies, Nash Equilibrium
- Selfish Caching
- (Optimistic) Price of Anarchy
- Braess' Paradox
- Rock-Paper-Scissors
- Mechanism Design

Prisoner's Dilemma

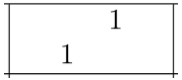
- Both u and v have a better combined outcome if they cooperate – but individually, defecting always yields a better result

v \ u		Player u	
		Cooperate	Defect
Player v	Cooperate	1 1	0 3
	Defect	3 0	2 2

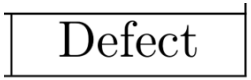
Table 25.1: The prisoner's dilemma game as a matrix.

Terminology

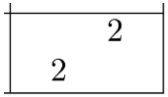
- Social optimum: Minimizes global costs



- Dominant strategy: Individually never worse than another strategy



- Nash Equilibrium: No-one can improve by changing only their own strategy



If every player plays a dominant strategy, we have a Nash Equilibrium

Prisoner's Dilemma - Terminology

Systems@ETH Zürich

Distributed Computing

Strategy of player v: Player v will play "Defect"

Strategy Profile: Player v will play "Defect" and player u will play "Cooperate"

Dominant Strategy: Playing "Defect"

Social Optimum: Both players cooperate

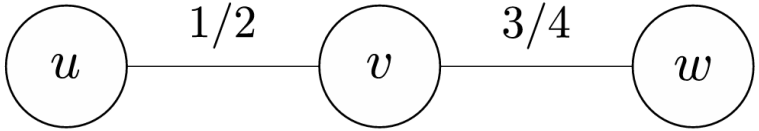
Nash Equilibrium: Both players defect

v \ u		Player u	
		Cooperate	Defect
Player v	Cooperate	1 1	0 3
	Defect	3 0	2 2

Table 25.1: The prisoner's dilemma game as a matrix.

Selfish caching

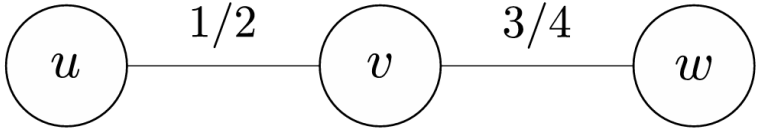
Consider a network where a node can either cache a file or fetch it from another node within the network. Storing a file costs 1, the demand for it is defined by d (in many examples defined as $d_u = d_v = d_w = 1$), while fetching a file costs $d * (shortest\ path\ to\ file)$.



$$c_{u \leftarrow v} = \frac{1}{2},\ c_{u \leftarrow w} = \frac{1}{2} + \frac{3}{4} = 1.25$$

Nash Equilibrium: Two exist, either u and w store the file, or v stores the file

(Optimistic) Price of Anarchy



If v stores the file, we have total cost of

$$NE_+ = d_v + c_{u \leftarrow v} + c_{w \leftarrow v} = 1 + \frac{1}{2} + \frac{3}{4} = 2.25$$

If u and w store the file, we have total cost of

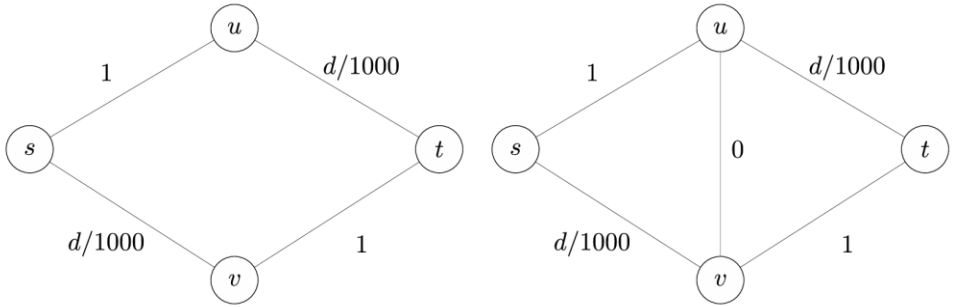
$$NE_- = d_u + d_w + c_{v \leftarrow u} = 1 + 1 + \frac{1}{2} = 2.5$$

The first strategy is also the socially optimal strategy. Therefore, the Price of Anarchy is $PoA = \frac{NE_-}{SO} = \frac{2.5}{2.25} \approx 1.11$

Since a Nash Equilibrium which is also the Social Optimum exists, the Optimistic Price of Anarchy is $OPoA = \frac{NE_+}{SO} = \frac{2.25}{2.25} = 1$

Braess' Paradox

Adding more paths can increase the costs



Assume we have 1000 drivers traveling from s to t. Without the path from u to v, they would split up $50/50$ with total costs for each being 1.5. Adding the path would mean all drivers take the same path, increasing their cost to 2.

Rock-Paper-Scissors

- Has no Nash Equilibrium

v \ u		Player u		
		Rock	Paper	Scissors
Player v	Rock	0 0	1 -1	-1 1
	Paper	-1 1	0 0	1 -1
	Scissors	1 -1	-1 1	0 0

Mixed Nash Equilibrium

Systems@ETH Zürich

Distributed Computing

Definition 25.16 (Mixed Nash Equilibrium)**.** *A Mixed Nash Equilibrium (MNE) is a strategy profile in which at least one player is playing a randomized strategy (choose strategy profiles according to probabilities), and no player can improve their expected payoff by unilaterally changing their (randomized) strategy.*

Theorem 25.17. *Every game has a mixed Nash Equilibrium.*

Rock-Paper-Scissors

- Has no Nash Equilibrium

v \ u		Player u Rock	Player u Paper	Player u Scissors
Player v	Rock	0 0	1 -1	-1 1
	Paper	-1 1	0 0	1 -1
	Scissors	1 -1	-1 1	0 0

- New definition: Mixed Nash Equilibrium – every game has one
- Reached by choosing each with probability $\frac{1}{3}$

Mechanism Design

Instead of analysing an existing system, we try to create one that incentivises nodes to behave nicely.

- Auction: One good is sold to one bidder
- First price auction: One good is sold to the highest bidder for his bid
- Truthful auction: No player can gain anything by lying

First price auctions are not truthful, since the highest bidder could decrease his bid to $b_1 - \varepsilon > b_2$ to save money.

- Second price auction: One good is sold to the highest bidder for the second highest bid

Second price auctions are truthful

Remarks

- Mechanism design assumes players want to maximise their profit. In the real world, they might not only be selfish, but also byzantine.
- Costs in selfish-caching can also be defined to be negative, nodes that cache could be rewarded for caching.

Eventual Consistency & Bitcoin

Systems@ETH Zürich

Distributed Computing

- Consistency, Availability, and Partitions
- Weak Consistency
- Bitcoin
- Layer 2 (Smart Contracts)
- Selfish Mining

Consistency, Availability, and Partition Tolerance

Systems@ETH Zürich

Distributed Computing

- Consistency:
  - All nodes agree on the current state of the system
- Availability:
  - The system is operational and instantly processing incoming requests
- Partition tolerance:
  - Still works correctly if a network partition happens
- Good news:
  - achieving any two is very easy
- Bad news:
  - achieving three is impossible (CAP theorem)
- => Eventual Consistency:
  - Guarantees that the state is eventually agreed upon, but the nodes may disagree temporarily

Weak Consistency

Systems@ETH Zürich

Distributed Computing

Eventual Consistency is only one form of Weak Consistency

- Monotonic Read Consistency: If a node u has seen a particular value of an object, any subsequent access of u will never return any older values
- Monotonic Write Consistency: A write operation of a node is completed before any successive write operation by the same node
- Read-Your-Write Consistency: After a node u has updated a data item, any later read from that node u will never see an older value
- Causal Consistency: Need the definition of Causal Relation first

Causal Relation

Systems@ETH Zürich

Distributed Computing

Definition 26.11 (Causal Relation)**.** *The following pairs of operations are said to be causally related:*

- *Two writes by the same node to different variables.*
- *A read followed by a write of the same node.*
- *A read that returns the value of a write from any node.*
- *Two operations that are transitively related according to the above conditions.*

Weak Consistency

Systems@ETH Zürich

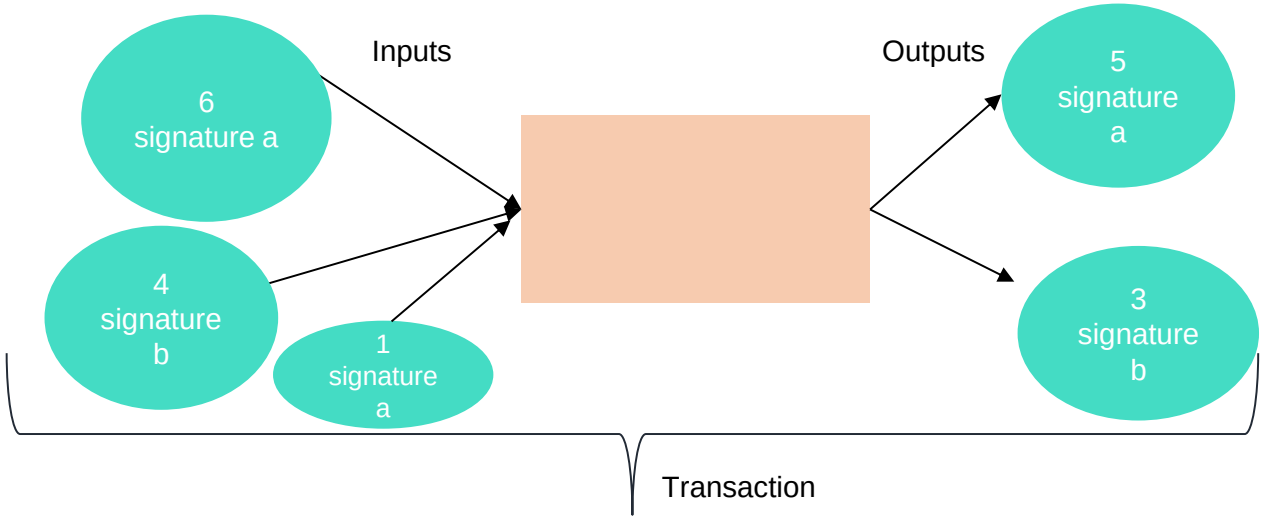
Distributed Computing

Eventual Consistency is only one form of Weak Consistency

- Monotonic Read Consistency: If a node u has seen a particular value of an object, any subsequent access of u will never return any older values
- Monotonic Write Consistency: A write operation of a node is completed before any successive write operation by the same node
- Read-Your-Write Consistency: After a node u has updated a data item, any later read from that node u will never see an older value
- Causal Consistency: A system provides causal consistency if operations that potentially are causally related are seen by every node of the system in the same order. Concurrent writes are not causally related and may be seen in different orders by different nodes.

Bitcoin

- Decentralized network consisting of nodes
- Users generate private/public key pair
  - Address is generated from public key
  - It is difficult to get users “real” identity from public key



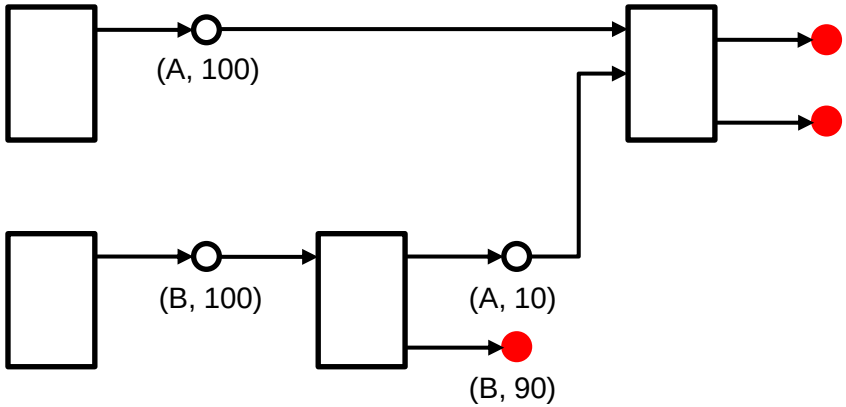
Bitcoin Transactions

Systems@ETH Zürich

Distributed Computing

- Conditions:
  - Sum of inputs must always be at least the sum of outputs
    - Unused part is used as transaction fee, gets paid to miner of block
  - An input must always be some whole output, no splitting allowed!
  - Money that a user “has” is defined as sum of unspent outputs

Bitcoin Transactions

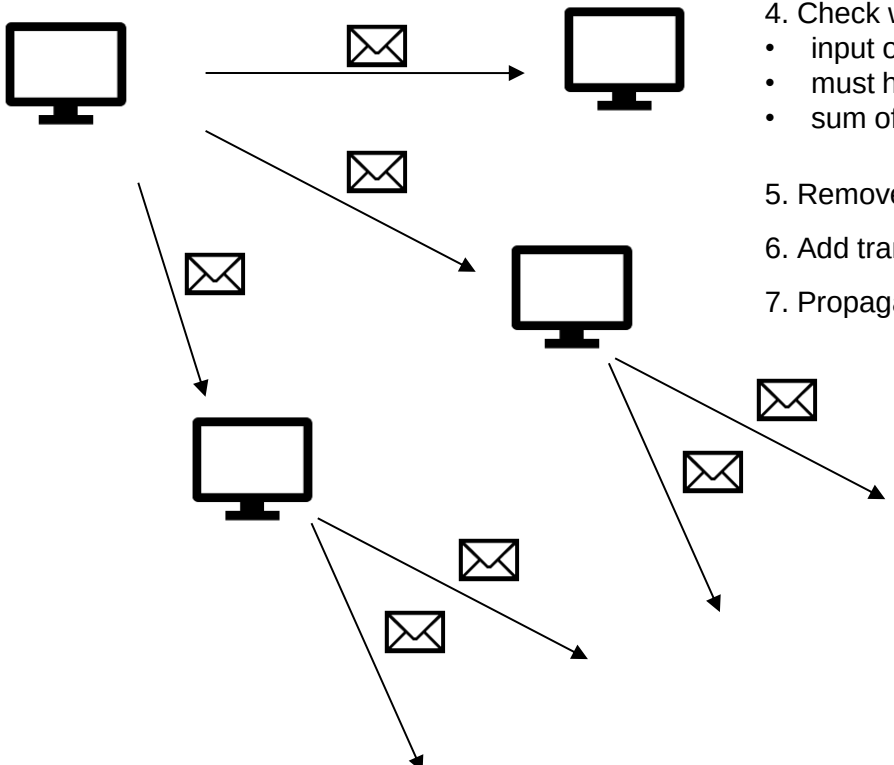


Set of unspent transaction outputs (UTXOs):
- This set is the shared state of Bitcoin
- The red outputs

Transaction Broadcast

1. Issue transaction
2. Add transaction to local history

3. Send transaction to other nodes in network



4. Check whether transaction is valid
   - input of transaction must be in local UTXO
   - must have valid signature
   - sum of inputs >= sum of outputs

5. Remove any input of transaction from local UTXO

6. Add transaction to local history

7. Propagate transaction further

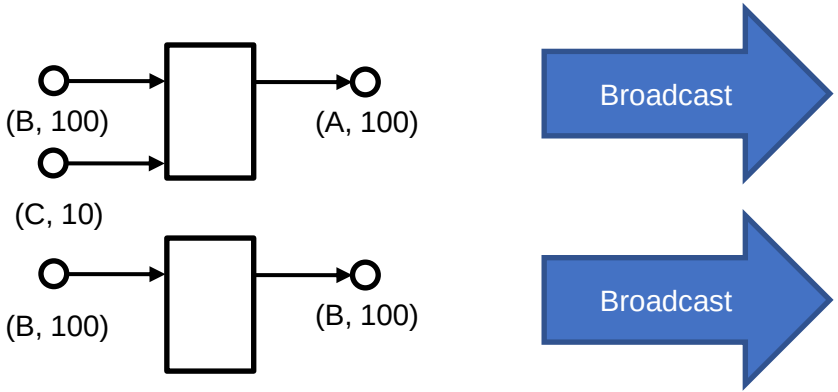
Doublespend Attack

Systems@ETH Zürich

Distributed Computing

- Multiple transactions attempt to spend the same output

- Ex: In a transaction, an attacker pretends to transfer an output to a victim, only to doublespend the same amount in another transaction back to itself.



Proof-of-Work

- Right now we have infinitely growing memory pool and we can't be sure that other nodes have the same pool
- Solution: Propagate memory pool through network and make sure everybody else will have same state
- Problem: How to avoid that everybody wants to propagate its own memory pool?
- Solution: Proof-of-Work
  - Proof that you put a certain amount of work into propagating your memory pool

Proof-of-Work

Systems@ETH Zürich

Distributed Computing

- Mining Blocks requires to proof that a certain amount of computational resources has been utilized

$$F_d(c, x) \rightarrow \{true, false\}$$

  - d: difficulty (is adapted all 24h)
  - c: challenge (the transactions and the hash of the previous block)
  - x: nonce (has to be found)

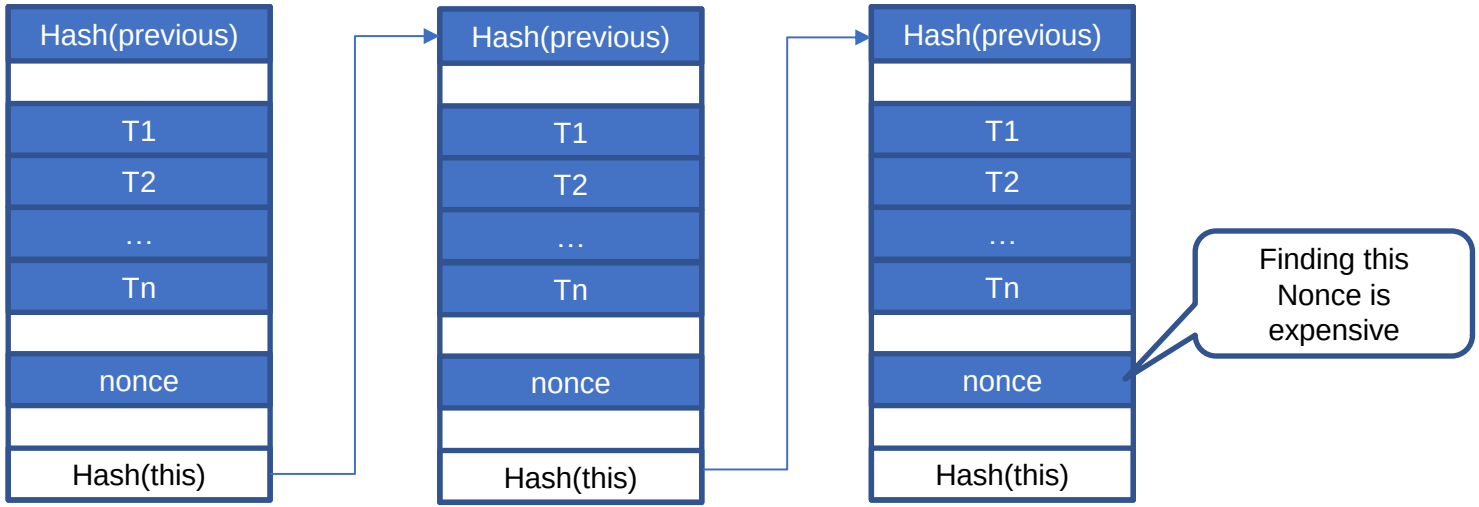
  - For fixed parameters d and c, finding x such that the function

$$\mathcal{F}_d(c, x) \rightarrow \mathrm{SHA256}(\mathrm{SHA256}(c|x)) < \frac{2^{224}}{d}.$$

Bitcoin chooses the difficulty such that a block is created all ~10 min

Block

- Data structure holding transactions reference to previous blocks and a nonce.
  - Header also contains more fields, such as a timestamp, the difficulty, network version, etc.
- Miner creates blocks with transactions from its memory pool



Mining

Systems@ETH Zürich

Distributed Computing

- Why should someone mine blocks?
  - You get a reward for each block you mine
  - You get the fee in the transactions

Bitcoin:

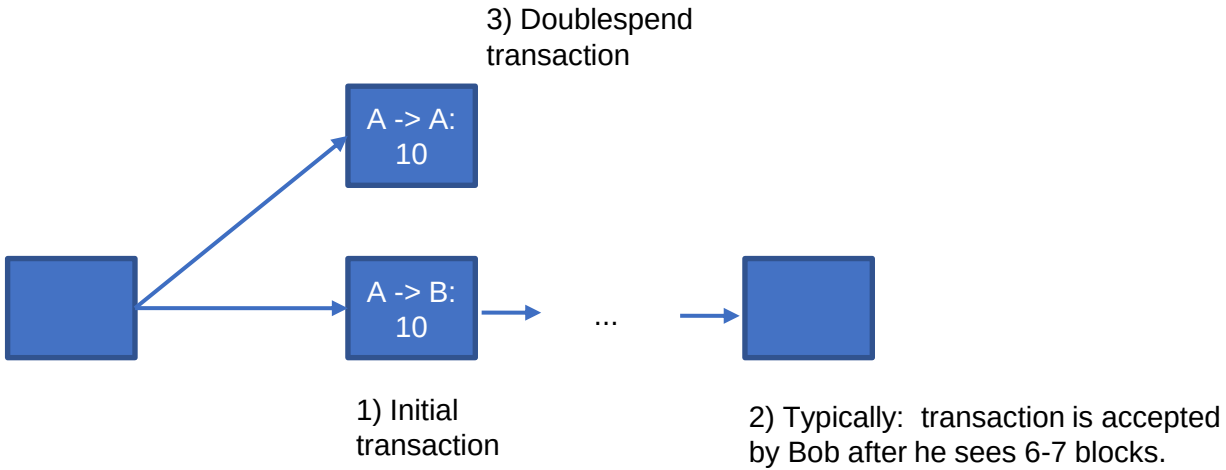
- Reward started at 50B and it is being halved every 210,000 blocks or 4 years in expectation
- This bounds the total number of Bitcoins to 21 million
- What will happen after that?

- Fee is the positive difference of input-output
- Miner include transactions which have a high fee.

- Problem: More miners -> more blocks are mined -> higher difficulty -> more Power needed

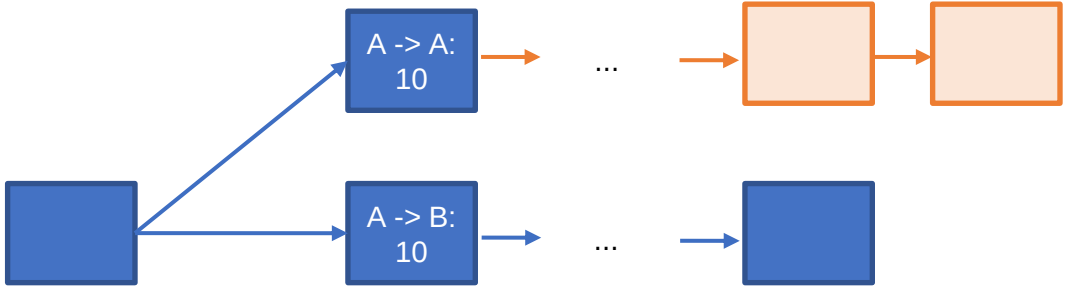
How does this prevent double spending?

- An intruder needs to have more than 50% of computation power to be faster in mining than all other together



How does this prevent double spending?

- An intruder needs to have more than 50% of computation power to be faster in mining than all other together



3) Doublespend transaction

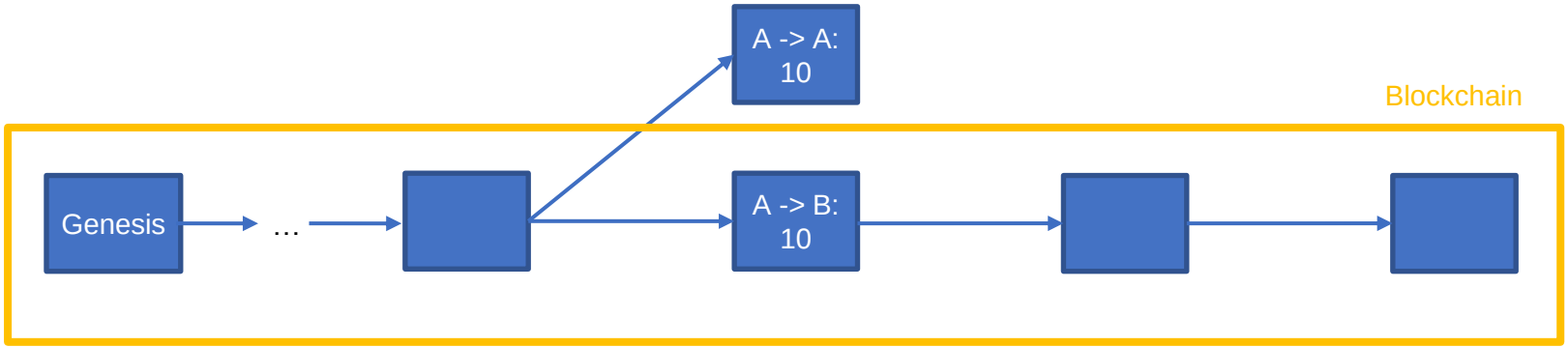
4) The goal of Alice is now to make the branch where she spends the money to herself growing faster.

1) Initial transaction

2) Typically: The transaction is accepted by Bob after he sees 6-7 blocks.

Blockchain

- Starts with the genesis block and is the longest path from this genesis block to a leaf.
- Consistent transaction history on which all nodes eventually agree



Note: To ensure that you'll get the money you should wait 5-10 further blocks

Smart Contracts

Systems@ETH Zürich

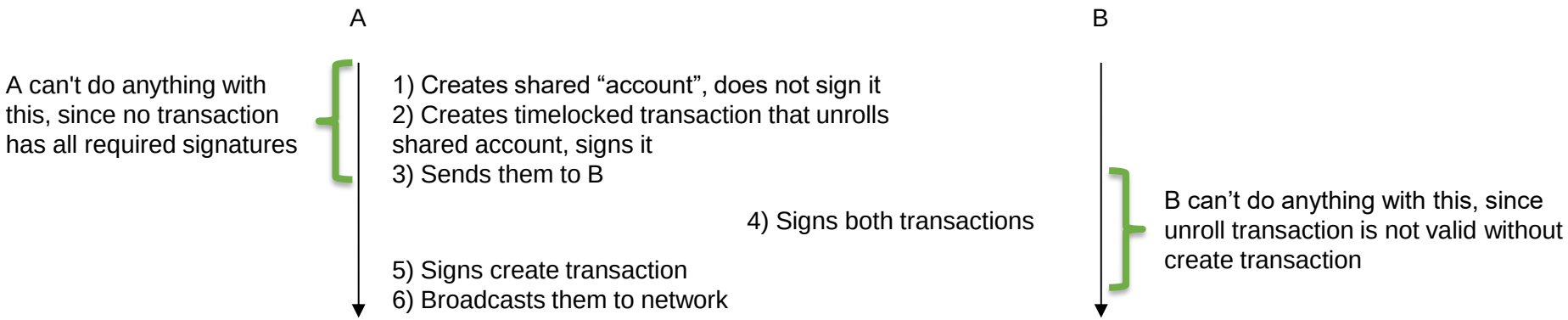
Distributed Computing

- Contract between two or more parties, encoded in such a way that correct execution is guaranteed by blockchain
  - Timelock transaction: Tx will only get added to memory pool after some time has expired
  - Micropayment channel:
    - Idea: Two parties want to do multiple small transactions, but want to avoid fees. So they only submit first and last transaction to blockchain and privately do everything in between

Micropayment Channel Setup Transaction

Algorithm 26.32 Parties A and B create a 2-of-2 multisig output o

1: B sends a list I_B of inputs with c_B coins to A
2: A selects its own inputs I_A with c_A coins
3: A creates transaction $t_s\{[I_A, I_B], [o = c_A + c_B \rightarrow (A, B)]\}$
4: A creates timelocked transaction $t_r\{[o], [c_A \rightarrow A, c_B \rightarrow B]\}$ and signs it
5: A sends t_s and t_r to B
6: B signs both t_s and t_r and sends them to A
7: A signs t_s and broadcasts it to the Bitcoin network



Micropayment Channel

Algorithm 26.33 Simple Micropayment Channel from s to r with capacity c

1: $c_s = c$, $c_r = 0$
2: s and r use Algorithm 26.32 to set up output o with value c from s
3: Create settlement transaction $t_f\{[o], [c_s \to s, c_r \to r]\}$
4: **while** channel open **and** $c_r < c$ **do**
5: In exchange for good with value δ
6: $c_r = c_r + \delta$
7: $c_s = c_s - \delta$
8: Update t_f with outputs $[c_r \to r, c_s \to s]$
9: s signs and sends t_f to r
10: **end while**
11: r signs last t_f and broadcasts it

Set up shared account and unrolling

Create settlement transaction

While buyer still has money and timelock not expired

Exchange goods and adapt money

Update settlement transactions with new values

S signs transaction and sends it to R

R signs last transaction and broadcasts it before timelock expires

Why does s sign it?

- Like this, R always holds all fully signed transactions and can choose the last one (where he gets the most money)
- S cannot submit any transaction, so S cannot get the goods and later submit a transaction where S did not pay the money for it

Selfish Mining

Systems@ETH Zürich

Distributed Computing

Selfish Mining: A selfish miner hopes to earn a reward of a larger share of blocks than its hardware would allow. He achieves this by temporarily keeping newfound blocks.

- Selfish mining can become the dominant strategy depending on how the network is set up
- Depends on γ, the share of altruistic miners that the selfish miner can reach before they receive the new correct block, and α, the share of computational power of the selfish miner
- If $\gamma=0$, the selfish miner needs $1/3$ of the computational power
- If $\gamma=1/2$, having $1/4$ of the computational power is enough
- If $\gamma=1$, selfish mining is always dominant

Quiz – Game Theory

Alice and Carol went on a trip and bought two identical antique vases for 500 Francs each. Unfortunately, their luggage was not handled properly when flying back home, and both vases broke. The airport manager wants to compensate Alice and Carol for the broken vases. As he does not know the true price of such a vase, he proposes a scheme.
Alice and Carol are placed in separate rooms so that they cannot communicate. Each of them is asked to write down the price of one vase as a natural number between 20 and 1000. Then, let p_{Alice} and p_{Carol} denote the prices they wrote and let Δ denote an integer ($0 \leq \Delta \leq 20$). They will be compensated as follows:

- If $p_{Alice} = p_{Carol} = p$, then p must be the true price of the vase and Alice and Carol receive p Francs each.
- If $p_{Alice} > p_{Carol}$, the airport manager assumes that Alice lied and Carol's price is correct. Then, Alice receives $p_{Carol} - \Delta$ Francs, and Carol receives $p_{Carol} + \Delta$ Francs.
- Symmetrically, if $p_{Alice} < p_{Carol}$, the airport manager assumes that Carol lied and Alice's price is correct. Then, Alice receives $p_{Alice} + \Delta$ Francs, and Carol receives $p_{Alice} - \Delta$ Francs.

Alice and Carol are both smart and rational. Each of them wants to maximize their own compensation, even if it means to lie to the airport manager.

a) [3] Is writing down the true price of the vase a dominant strategy for Alice when $\Delta = 5$? Why?

No.

Let's say $p_{true} = 500$ and $p_{Carol} = 499$.
Here, writing down 500 yields 494, while writing down 498 yields 503

Quiz – Game Theory

Alice and Carol went on a trip and bought two identical antique vases for 500 Francs each. Unfortunately, their luggage was not handled properly when flying back home, and both vases broke. The airport manager wants to compensate Alice and Carol for the broken vases. As he does not know the true price of such a vase, he proposes a scheme.
Alice and Carol are placed in separate rooms so that they cannot communicate. Each of them is asked to write down the price of one vase as a natural number between 20 and 1000. Then, let p_{Alice} and p_{Carol} denote the prices they wrote and let Δ denote an integer ($0 \leq \Delta \leq 20$). They will be compensated as follows:

- If $p_{Alice} = p_{Carol} = p$, then p must be the true price of the vase and Alice and Carol receive p Francs each.
- If $p_{Alice} > p_{Carol}$, the airport manager assumes that Alice lied and Carol's price is correct. Then, Alice receives $p_{Carol} - \Delta$ Francs, and Carol receives $p_{Carol} + \Delta$ Francs.
- Symmetrically, if $p_{Alice} < p_{Carol}$, the airport manager assumes that Carol lied and Alice's price is correct. Then, Alice receives $p_{Alice} + \Delta$ Francs, and Carol receives $p_{Alice} - \Delta$ Francs.

Alice and Carol are both smart and rational. Each of them wants to maximize their own compensation, even if it means to lie to the airport manager.

b) [3] Find a pure Nash Equilibrium for $\Delta = 5$.

For every value p that one of them chooses, the other is better of choosing $p - 1$. Therefore, the only NE is a $p_C = p_A = 20$

Quiz – Game Theory

Alice and Carol went on a trip and bought two identical antique vases for 500 Francs each. Unfortunately, their luggage was not handled properly when flying back home, and both vases broke. The airport manager wants to compensate Alice and Carol for the broken vases. As he does not know the true price of such a vase, he proposes a scheme.
Alice and Carol are placed in separate rooms so that they cannot communicate. Each of them is asked to write down the price of one vase as a natural number between 20 and 1000. Then, let p_{Alice} and p_{Carol} denote the prices they wrote and let Δ denote an integer ($0 \leq \Delta \leq 20$). They will be compensated as follows:

- If $p_{Alice} = p_{Carol} = p$, then p must be the true price of the vase and Alice and Carol receive p Francs each.
- If $p_{Alice} > p_{Carol}$, the airport manager assumes that Alice lied and Carol's price is correct. Then, Alice receives $p_{Carol} - \Delta$ Francs, and Carol receives $p_{Carol} + \Delta$ Francs.
- Symmetrically, if $p_{Alice} < p_{Carol}$, the airport manager assumes that Carol lied and Alice's price is correct. Then, Alice receives $p_{Alice} + \Delta$ Francs, and Carol receives $p_{Alice} - \Delta$ Francs.

Alice and Carol are both smart and rational. Each of them wants to maximize their own compensation, even if it means to lie to the airport manager.

c) [5] What is the Optimistic Price of Anarchy ($OPoA$) for $\Delta = 1$?

Here, we have a NE for every $p_C = p_A$, so $NE_+ = 2000$. Since $SO = 2000$, we have $OPoA = \frac{2000}{2000} = 1$

Quiz – Game Theory

Alice and Carol went on a trip and bought two identical antique vases for 500 Francs each. Unfortunately, their luggage was not handled properly when flying back home, and both vases broke. The airport manager wants to compensate Alice and Carol for the broken vases. As he does not know the true price of such a vase, he proposes a scheme.
Alice and Carol are placed in separate rooms so that they cannot communicate. Each of them is asked to write down the price of one vase as a natural number between 20 and 1000. Then, let p_{Alice} and p_{Carol} denote the prices they wrote and let Δ denote an integer $(0 \leq \Delta \leq 20)$. They will be compensated as follows:

- If $p_{Alice} = p_{Carol} = p$, then p must be the true price of the vase and Alice and Carol receive p Francs each.
- If $p_{Alice} > p_{Carol}$, the airport manager assumes that Alice lied and Carol's price is correct. Then, Alice receives $p_{Carol} - \Delta$ Francs, and Carol receives $p_{Carol} + \Delta$ Francs.
- Symmetrically, if $p_{Alice} < p_{Carol}$, the airport manager assumes that Carol lied and Alice's price is correct. Then, Alice receives $p_{Alice} + \Delta$ Francs, and Carol receives $p_{Alice} - \Delta$ Francs.

Alice and Carol are both smart and rational. Each of them wants to maximize their own compensation, even if it means to lie to the airport manager.

d) [5] For which values of Δ is there a unique pure Nash equilibrium? Justify your answer.

For $2 \leq \Delta \leq 20$, we only have the NE $p_C = p_A = 20$. As seen in c), for $1 = \Delta$ we have 981 NEs. For $0 = \Delta$, we have the same 981 NEs, so $2 \leq \Delta \leq 20$.

Quiz

Systems@ETH Zürich

Distributed Computing

1.1 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?

1.1 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).

Quiz

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?

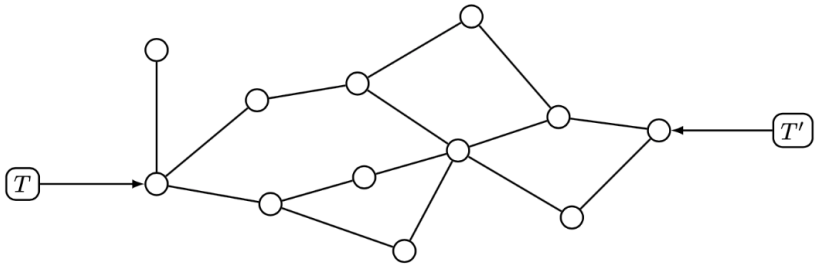


Figure 1: Random Bitcoin network

Quiz

Systems@ETH Zürich

Distributed Computing

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?

$$\frac{7}{12}$$

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?

$$20+4*\frac{80}{11} = 49\%$$

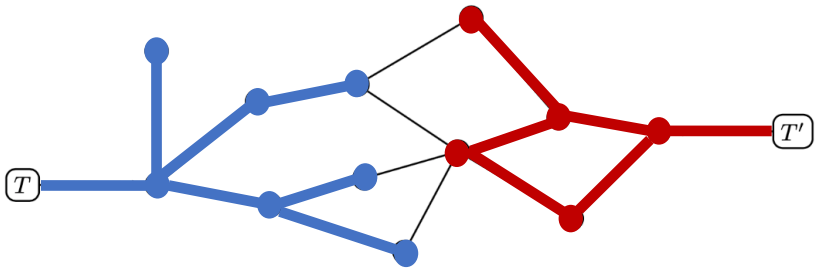


Figure 1: Random Bitcoin network