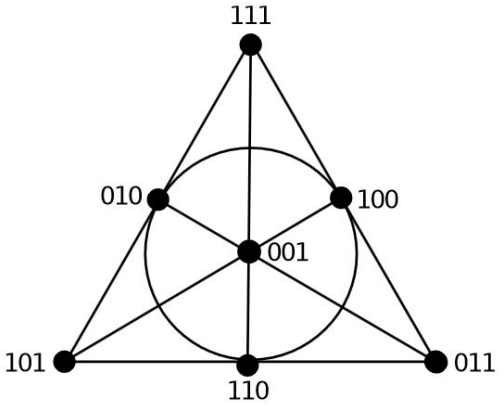
Computer Systems

Week 12

Last Exercise

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.



Last Exercise

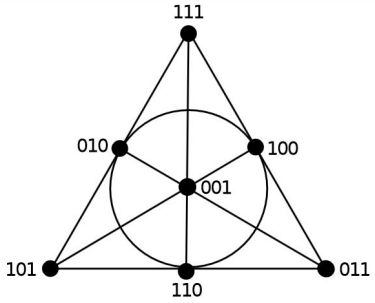


Figure 1: Quorum System

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

a) This quorum system consists of 7 quorums. As work is defined as the minimum (over all access strategies) expected number of servers in an accessed quorum, this system's work is 3 (all strategies induce the same work on a system where all quorums are the same size). The best access strategy consists of uniformly accessing each quorum (you will prove this for a more general case in exercise 3), so its load is 3/7.

Last Exercise

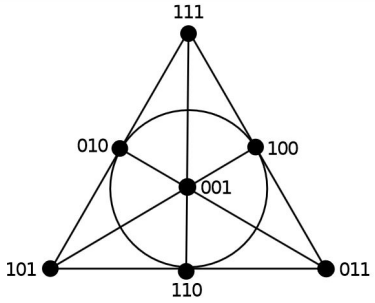


Figure 1: Quorum System

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

b) Its resilience $R(\mathcal{S}) = 2$. Proof: every node is in exactly 3 quorums, so 2 nodes can be contained in at most $2 \cdot 3 = 6 < 7 = |\mathcal{S}|$ quorums, thus if no more than 2 nodes fail, there will be at least 1 quorum without a faulty node. If on the other hand for example the nodes 101, 010 and 111 fail, no other quorum can be achieved; see also exercise 1a).

Last Exercise

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.

a) In an s-uniform quorum system each quorum has exactly s elements, so independently of which quorum is accessed, s servers have to work. Summed up over all servers we reach a total load of s, which is the work of the quorum system. As the load induced by an access strategy is defined as the maximum load on any server, the best strategy is to evenly distribute this work on all servers.

Last Exercise

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

b) Let $V = \{v_1, v_2, ..., v_n\}$ be the set of servers and $\mathcal{S} = \{Q_1, Q_2, ..., Q_m\}$ an s-uniform quorum system on V. Let Z be an access strategy, thus it holds that: $\sum_{Q \in \mathcal{S}} P_Z(Q) = 1$. Furthermore let $L_Z(v_i) = \sum_{Q \in \mathcal{S}; v_i \in Q} P_Z(Q)$ be the load of server v_i induced by Z.

Then it holds that:

$$\sum_{v_i \in V} L_Z(v_i) = \sum_{v_i \in V} \sum_{Q \in \mathcal{S}; v_i \in Q} P_Z(Q) = \sum_{Q \in \mathcal{S}} \sum_{v_i \in Q} P_Z(Q)$$

$$= \sum_{Q \in \mathcal{S}} P_Z(Q) \sum_{v_i \in Q} 1 \overset{*}{=} \sum_{Q \in \mathcal{S}} P_Z(Q) \cdot s = s \cdot \sum_{Q \in \mathcal{S}} P_Z(Q) = s$$

The transformation marked with an asterisk uses the uniformity of the quorum system.

To minimize the maximal load on any server, the optimal strategy is to evenly distribute this load on all servers. Thus if a balanced access strategy exists, this leads to a system load of s/n.

Last Exercise

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?

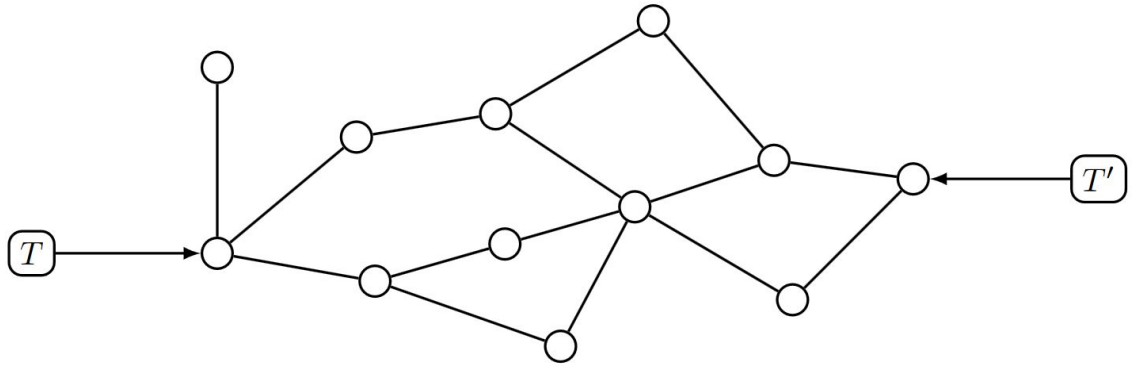


Figure 1: Random Bitcoin network

Last Exercise

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?

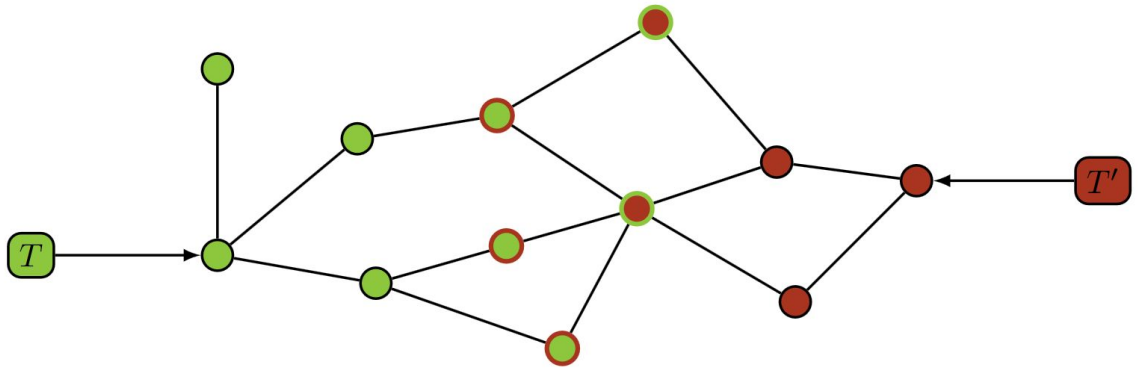


Figure 2: Random Bitcoin network

Last Exercise

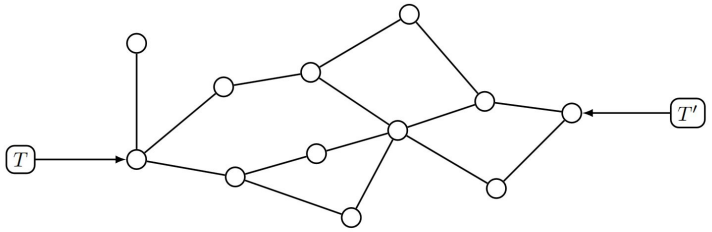


Figure 1: Random Bitcoin network

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.

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?

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.

Last Exercise

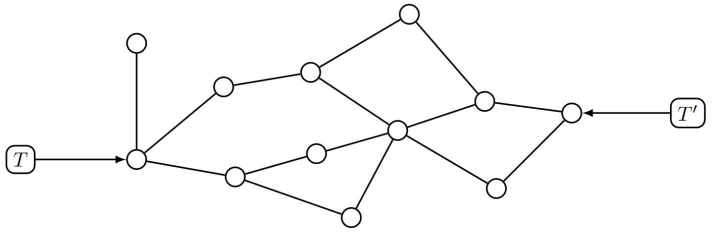


Figure 1: Random Bitcoin network

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.

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?

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.

Last Exercise

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.

Last Exercise

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.

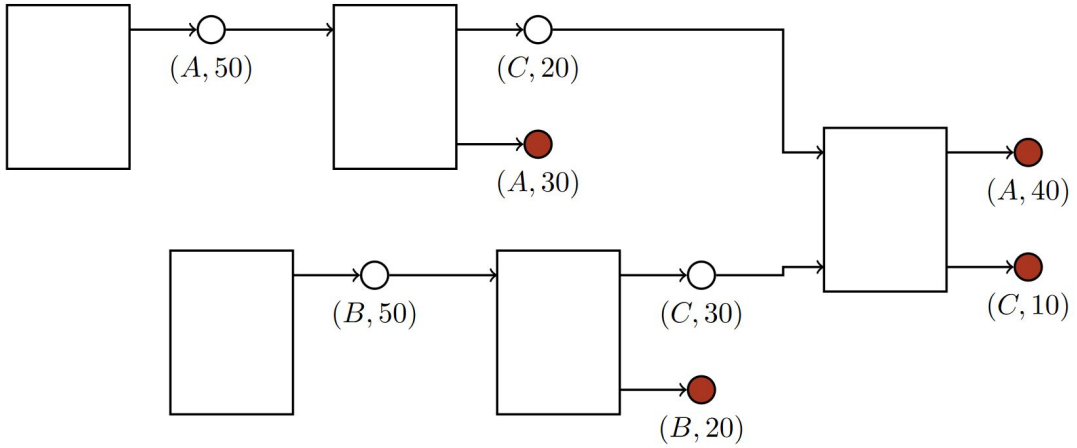


Figure 3: Transaction Graph. The red outputs are UTXOs.

Last Exercise

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.

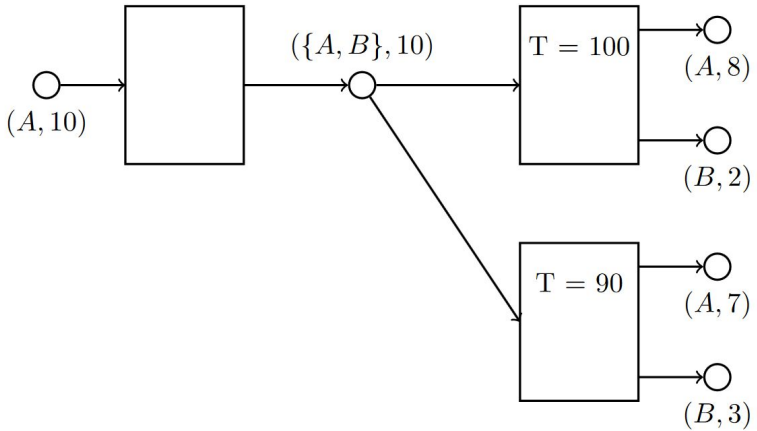
c) Fully spending an output simplifies the bookkeeping considerably as an output can only bein two possible states: spent or unspent. This means that it is easy to detect conflicts,because two transactions spending the same output are conflicts.

Last Exercise

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.

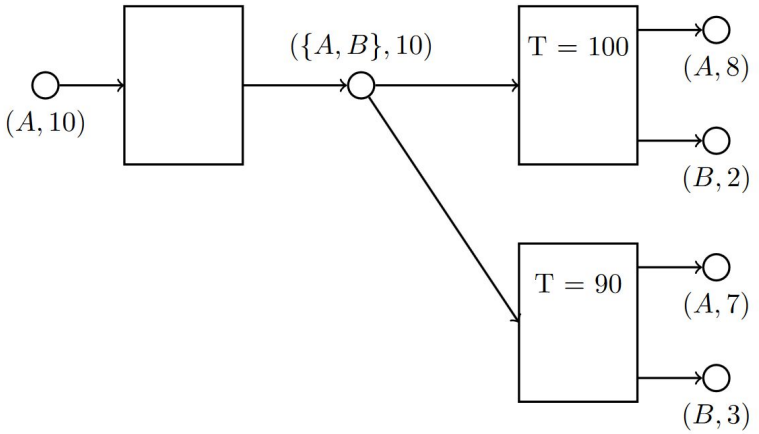
Building on this idea, a "payment channel"[1] can be created where money can be exchanged with someone else securely without doing every transaction on the blockchain. This works by replacing the transaction and moving money between the outputs. The construction is shown in Figure 2.



Last Exercise - 2.4 Bitcoin Script

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

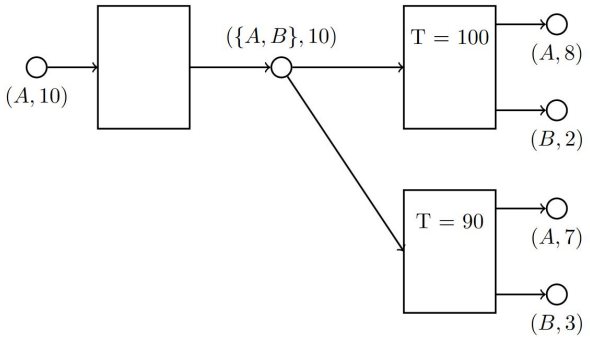
a) Transactions are instantly finalized, so the large confirmation delay of the blockchain is irrelevant. Only the signatures of both parties are needed, then the money has effectively changed the owner. Furthermore no transaction fees have to be paid to miners for replacing a transaction.



Last Exercise - 2.4 Bitcoin Script

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?

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.

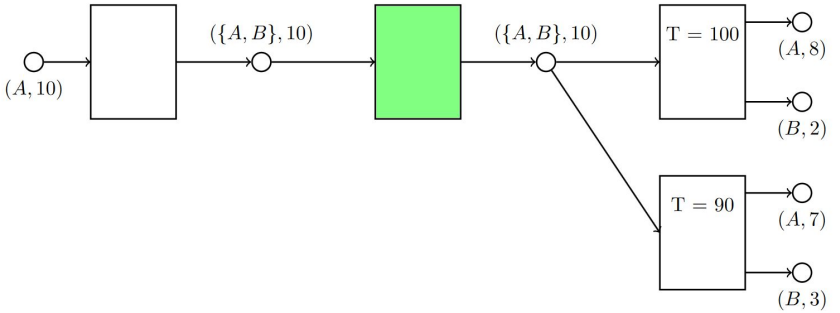


Last Exercise - 2.4 Bitcoin Script

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?

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.



Game Theory

What is Game Theory?

“Game theory is a sort of umbrella or ‘unified field’ theory for the rational side of social science, where ‘social’ is interpreted broadly, to include human as well as non-human players (computers, animals, plants).” – **Robert Aumann**, 1987

Or

The study of constructing mathematical models to analyze strategic interactions among people trying to make rational decisions.

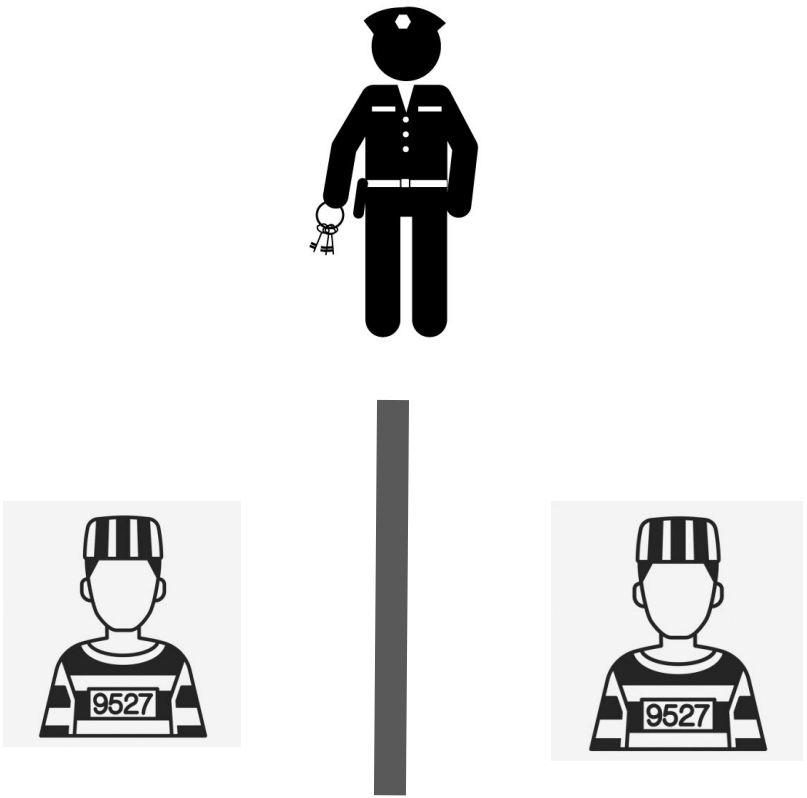
What is Game Theory?

Example: *Prisoner's Dilemma*

Two prisoners questioned by the police

Can cooperate with the other prisoner and remain silent or defect the other prisoner and talk.

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



Some Terminology

- **Strategy** - A sequence of moves for a game
- **Strategy Profile** - A possible outcome of a game
- **Social optimum (SO)** - Strategy profile with the best outcome for all players
- **Dominant strategy** - Player is never worse of when playing this strategy
- **Nash Equilibrium (NE)** - Strategy profile where no player can improve alone

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

(Optimistic) Price of Anarchy ((O)PoA)

NE_ : Nash Equilibrium with the highest cost

NE+: Nash Equilibrium with the smallest cost

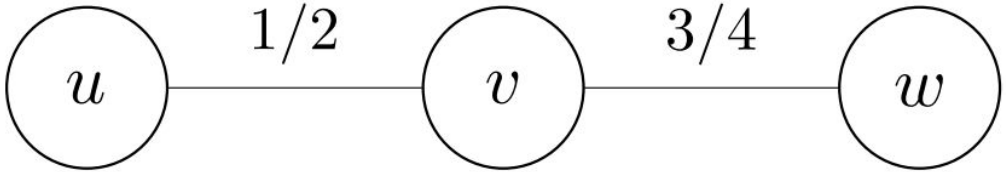
$$PoA = \frac{\text{cost}(NE\_)}{\text{cost}(SO)}$$

$$OPoA = \frac{\text{cost}(NE_+)}{\text{cost}(SO)}$$

Game Example: Selfish Caching

A node may cache or not cache a file

If it caches, the access cost will be 1. Else the cost is `((shortest path to cache) * demand)`



In this Example: Each node has demand 1.

Game Example: Rock, Paper, Scissors

No Nash Equilibrium. But there is a mixed Nash Equilibrium: Chose every strategy with probability of 1/3

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

Game Example: Among Us

A Crew on a spaceship wants to go home.
Problem: There is an Impostor among them who wants to kill everyone...

#Crewmembers > #Impostors
=> Life of Impostor is more valuable

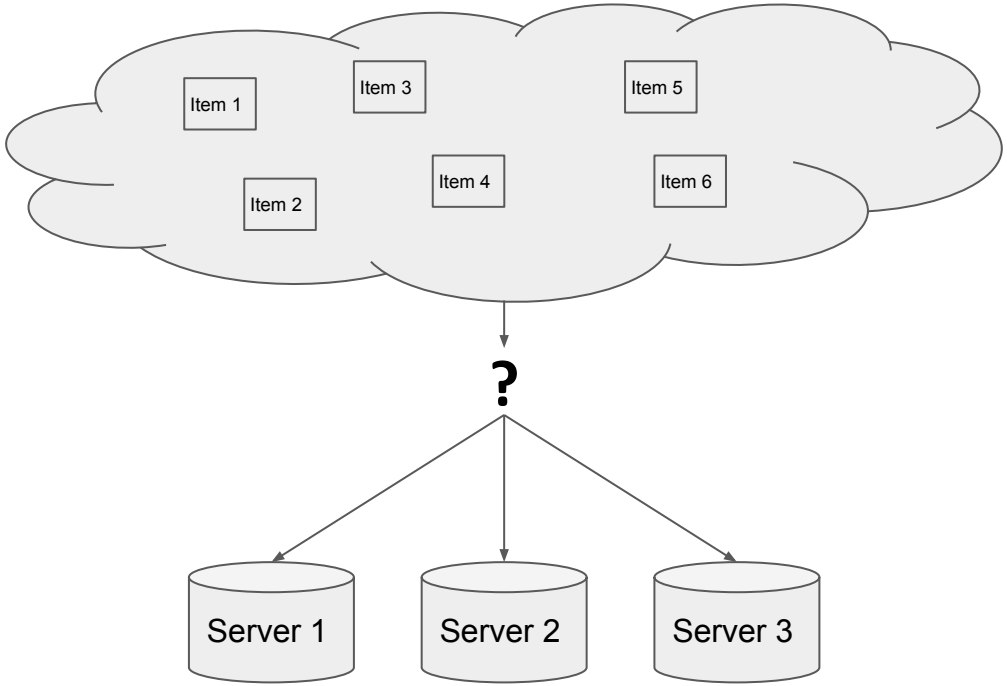
Crewmembers can choose to trade one of their members to one of the impostor. A win for the Crewmembers.

References:
Video from “The Game Theorists”:
https://www.youtube.com/embed/Ck604TT4hEg
Article “Among Us: A game theory analysis”:
https://medium.com/@anirudh.raj.iyengar/among-us-a-game-theory-analysis-b020454dc594
Article “Among Us and Game Theory”:
https://medium.com/@kaustubh.g/among-us-and-game-theory-f74c8ac9f05

Distributed Storage

Problem

How to store many items on many nodes in a “**consistent**” manner?



Consistent Hashing

Use **hash functions** to transform item *(x)* and node IDs *(v)* into values in **[0,1)**
Item is stored on machine with the closest hash

$$|h_i(x) - h(u)| = \min_v \{|h_i(x) - h(v)|\}, \text{ for any } i$$

Some properties of **consistent hashing**:

- Each node stores the same number of items in expectation
- Any single node's memory consumption is bounded (Fact 24.3, Chernoff bound)
- Supports nodes leaving/joining

Hypercubic Networks

In a classic distributed system one node can have a view of the entire system because nodes rarely leave/join

However, we are considering **very large networks** with **high churn** in which it becomes impossible for nodes to have an accurate and updated picture of large parts of the network topology

Thus, we want a system that only relies on every node knowing its **small neighborhood**

=> What kind of network topology should we use?.

Hypercubic Networks

Consistent hashing reminder: Where to store items.

Hypercubic networks: Arrange nodes such that they form a virtual network, also called an **overlay network**.

In general, the overlay network gives us the possibility to “navigate” our distributed storage system, i.e., do **routing**. This is necessary since each node only has a local view, but we still want to find any item, even if it is not in the neighborhood of the node we are currently querying.

Hypercubic Networks

A good overlay topology should fulfill the following properties (more or less)

- **Homogeneity**: No single point of failure, all nodes are “equal”
- **Node IDs in [0,1)** for consistent hashing
- Nodes have **small degree**, i.e., only relatively few neighbours
- **Small diameter** and easy routing: Any node should be reachable within reasonable time

Different Hypercubic Networks

Different overlay topologies make different trade-offs, for example:

Mesh: *M(m,d)* - A Mesh with m nodes in one dimension and a dimension of d
Torus: *T(m,d)* - A Mesh where you wrap around at the end of a row/column

A Mesh M(2,d) == T(2,d) is a d-dimensional Hypercube

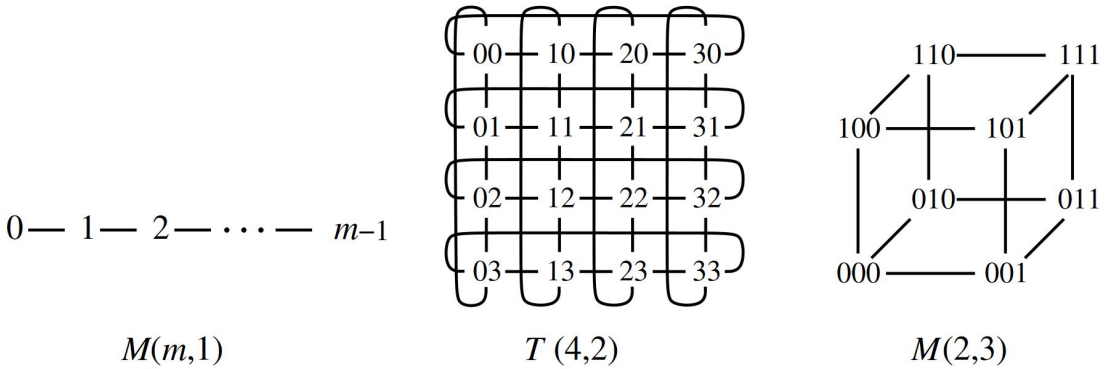


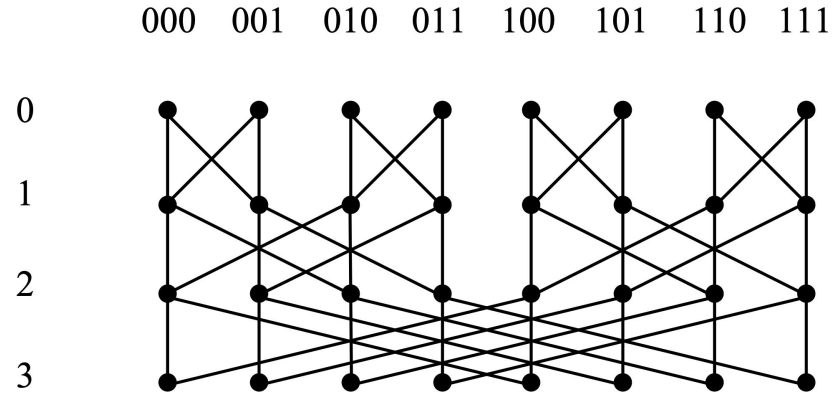
Figure 24.7: The structure of $M(m, 1)$, $T(4, 2)$, and $M(2, 3)$.

Different Hypercubic Networks

Different overlay topologies make different trade-offs, for example:

Butterflies: *BF(d)* - Constant small node degree

Shuffle-Exchange: *SE(d)* - Another constant degree network



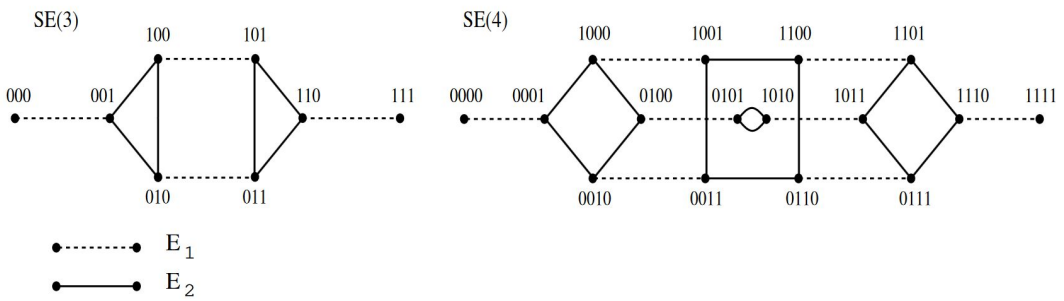


Figure 24.13: The structure of SE(3) and SE(4).

Distributed Hash Table

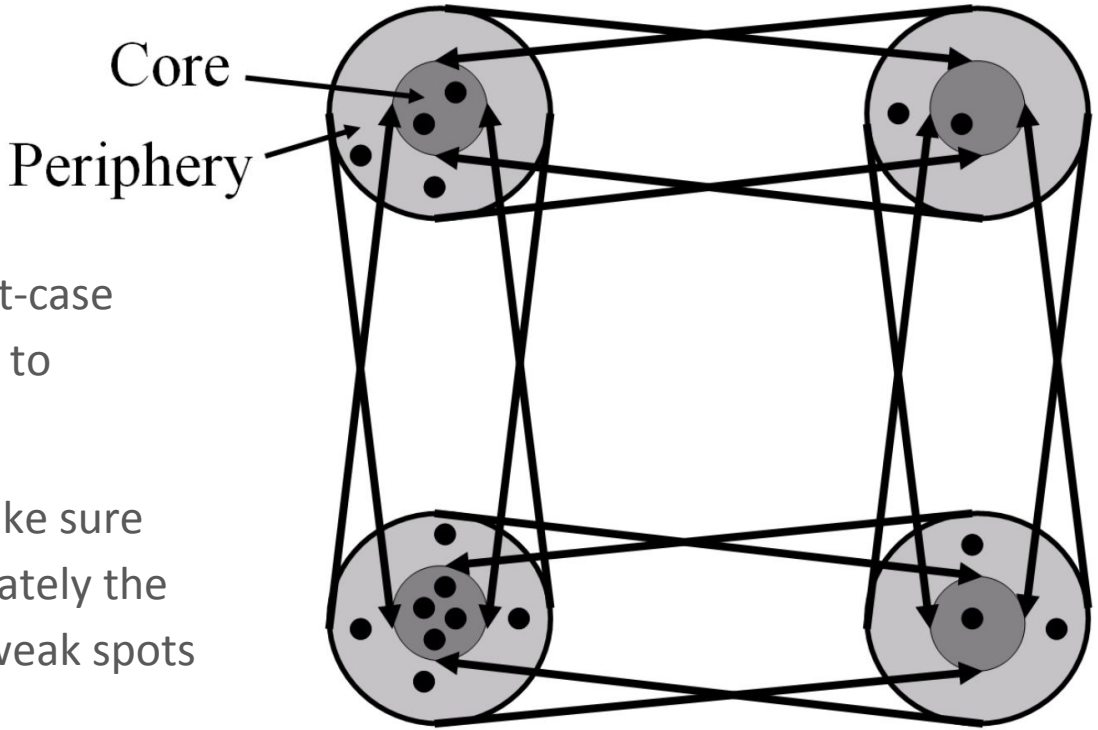
DHT: Distributed Hash Table

- Combines consistent hashing with overlay networks
- Supports searching, insertion and (maybe) deletion
- For example: Use hypercube with hyper nodes. “Core” nodes store data, “periphery” nodes can move around.

DHT & Churn

Robustness against Churn

- Attacker crashes nodes in worst-case manner. Can target weak spots to partition the DHT.
- DHT re-distributes nodes to make sure each hyper-node has approximately the same number of nodes => No weak spots



Quiz: 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?

Quiz

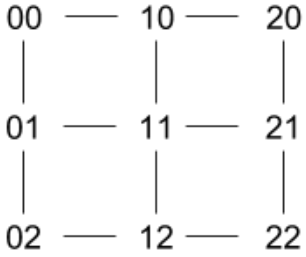
Draw the following hypercubic graphs:

- M(3,1)
- M(3,2)
- SE(2)
- M(2,4)

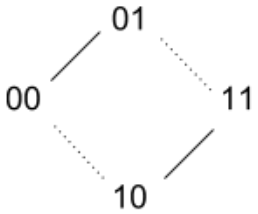
Quiz Solutions

M(3,1)

M(3,2)



SE(2)



M(2,4)