HS19

DISTRIBUTED SYSTEMS ADVANCED & BLOCKCHAIN

Classic Consensus, Broadcast, and Protocols

T. Bocek

thomas.bocek@hsr.ch

HSR
HOCHSCHULE FÜR TECHNIK
RAPPERSWIL

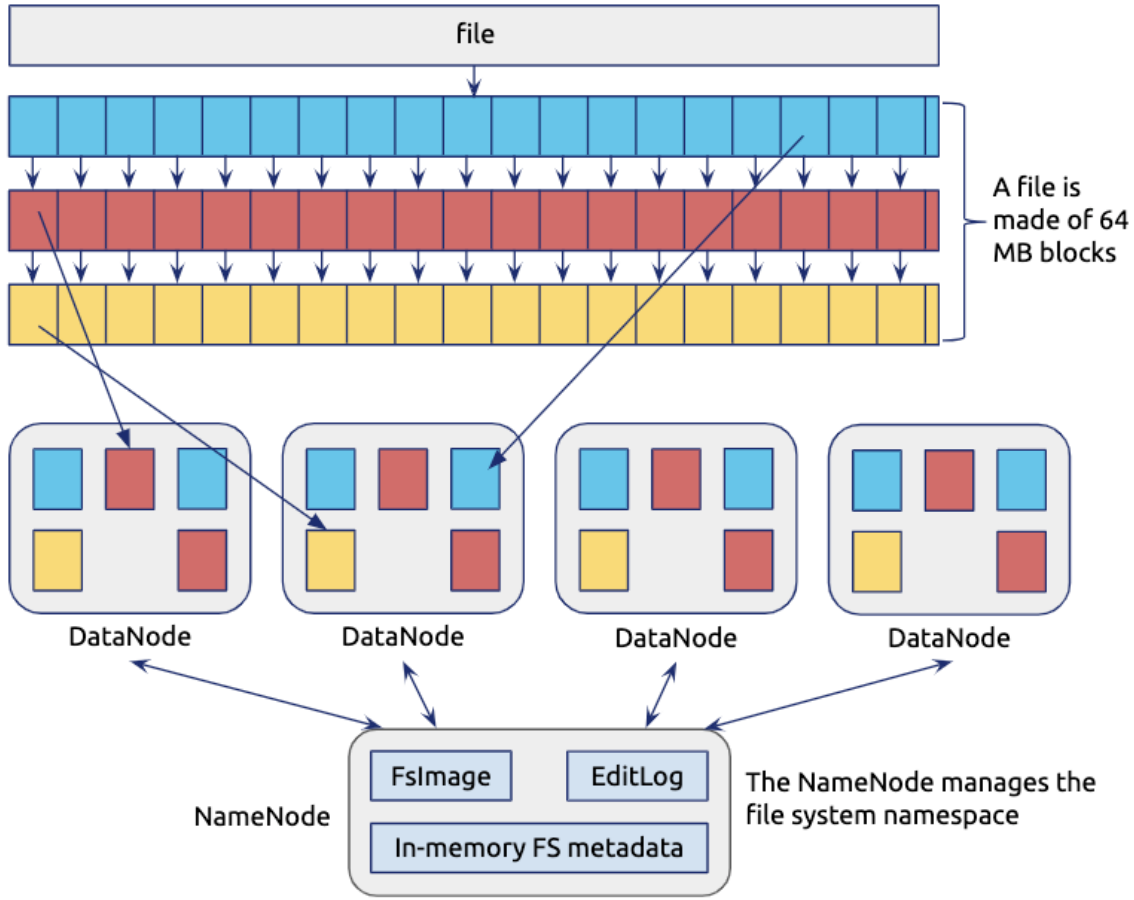
FHO Fachhochschule Ostschweiz

Distributed Systems - In the News

- **drand - A Distributed Randomness Beacon**
  - Write your own oracle
  - League of Entropy
    - “first time ever that a randomness beacon is run by several organizations”
    - “Every other existing randomness beacon is generated by an individual entity”, e.g. random.org
- **04.11.2019 Bitcoin (BTC) 2017 Rally Caused by Single Whale, Research Shows**
  - The series of $1,000 days that brought BTC to a record above $19,600 could have been the work of one entity
  - “price action was happening on a single exchange, and seemingly from a single account.”
  - Bloomberg: “The idea that a lone whale fueled a yearlong Bitcoin bull run is a little silly “
- **04.11.2019 Ethereum (ETH) Pushes Back Mining Freeze Once Again**
  - The Berlin hard fork will probably be used to code another delay for the mining ice age.
  - Q: ice age?
- **02.11.2019 Bitcoin, 11-years in**
  - Another point of view
- **05.11.2019 Facebook Libra is Architecturally Unsound**
  - “Libra’s byzantine tolerance on a permissioned network is an incoherent design.”
    - The algorithm chosen by Libra still has a worst-case $O(n^2)$ communication
  - TPS, Move language, Ed25519, consumer protection, reply

Distributed Systems - In the News

- **4.11.2019 Building a Large-scale Distributed Storage System Based on Raft**
  - TiKV, a large-scale open source distributed database based on Raft
  - Extended Raft with Regions
  - Github, ACID, Rust
  - Sharding: Data distribution of HDFS DataNode (see lecture 7, slide 36)
  - Other examples of consistent hashing?



http://energystudy.synergylabs.org/courses/15-440-Fall2017/lectures/16-gfs_hdfs_spanner.pdf

Distributed Systems - In the News

- **01.11.2019 DNS Wars**
  - DNS is fundamental part of the infrastructure of the Internet
  - Distributed database
  - Initial distribution of names was via flooding of a common copy of the hosts file
    - Scalability?
  - First DNS War was fought over the transition of DNS names in .com, .net and .org
    - Free → Network Solutions (monopoly)
    - Finally: competition in name registration services
  - Sitefinder
    - Wildcard for domain names that did not exist → redirect to search engine
- Akamai CDN, delivered result based on how close DNS resolver was
  - 8.8.8.8?
  - Extension mechanisms for DNS
    - "EDNS is also used for sending general information from resolvers to name servers about clients' geographic location in the form of the EDNS Client Subnet (ECS) option"
    - It destroys DNS privacy
- Today's DoH/DoT Wars
  - DNS over TLS/TCP
  - DNS over HTTPS
    - Why DoH at all? It doesn't appear to be solving a technology or a performance issue that is not already competently addressed in DoT – DoT/853 vs DoH/443, push DoH
  - Mozilla announced DoH to Cloudflare's Open Recursive Resolver

Challenge Task

- **Today deadline milestone**
  06.11.2018 - 23:59
  - 18 students on unterricht.hsr.ch, 14 students doing the challenge task on dsl.hsr.ch

- **Exam of last year online: [dsl.hsr.ch](dsl.hsr.ch)**

Aufgabe 5: Blockchains (12 Punkte)

Dieser Smart Contract hält in der Variable ***balances*** fest, welche Adresse wie viele Tokens hat. Die Funktion ***dividend()*** zahlt jeder Adresse einen Token aus.

```
pragma solidity ^0.4.25;

contract Exam {
    address private owner;
    mapping (address => uint256) private balances;
    address[] private balances_keys;

    constructor() {
        owner = msg.sender;
    }

    function dividend() public {
        require(owner == msg.sender);
        for(uint256 i=0;i<balances_keys.length;i++) {
            balances[balances_keys[i]] = balances[balances_keys[i]] + 1;
        }
    }
}
```

a) (4P) Was ist das Problem in der Funktion ***dividend()***

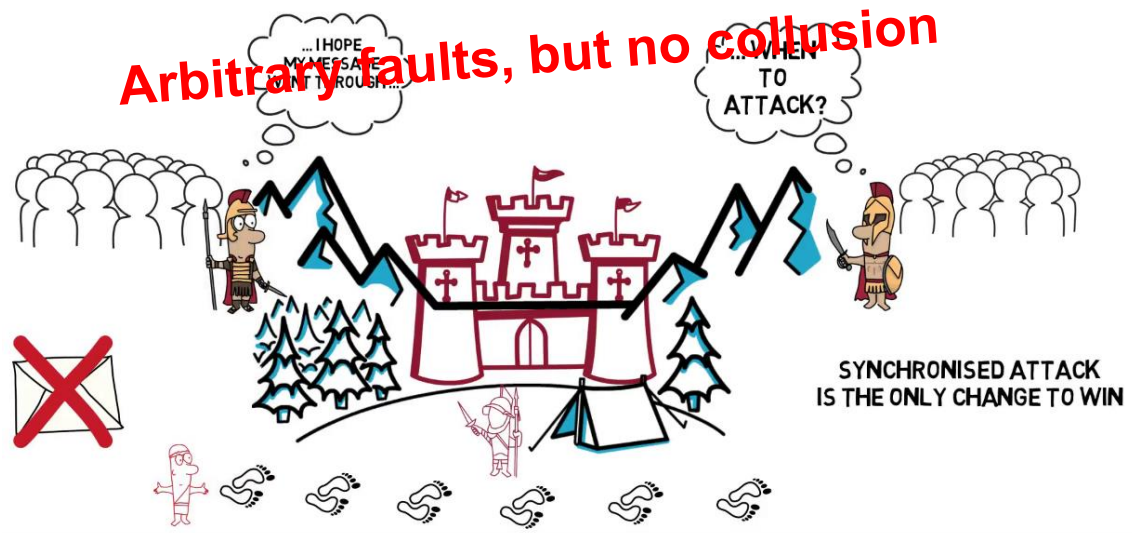
b) (4P) Bei einem ICO, welcher einen ERC-20 Token heraus gibt, könnte der Token direkt nach der Überweisung von Ethers an einem Smart Contract erstellt werden. Was ist der Nachteil für den Benutzer/Investor wenn man einen Token direkt erstellt?

c) (4P) Sie entwickeln eine eigene Blockchain mit PoW. Sie wissen, dass ein Miner 1.067 H/s leisten kann. Sie möchten, dass ein Miner in einer Minute alle Varianten mit der Nonce durchrechnen kann. Wie viele Bits muss die Nonce haben? (Hinweis: die Anzahl Bits ist ganzzahlig)

Consensus (last weeks lecture)

- **Definition: Consensus decision-making is a group decision-making process in which group members develop, and agree to support a decision in the best interest of the whole.**
- **A Byzantine fault is an arbitrary fault that occurs during the execution of an algorithm by a distributed system**
  - Not only crash, but lie or even collude to reach an advantage
- **Your own nodes, your control, no collusion**
- **Other forms of consensus**
  - Paxos
  - Raft
  - vDHT

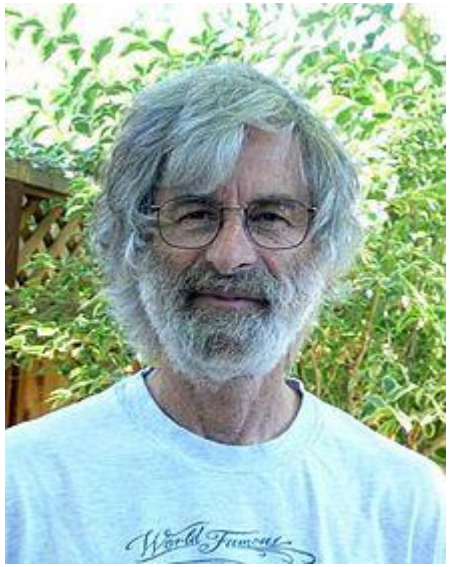
- **Often: consensus defines leader**
  - Leader creates block,
  - Leader adds data
  - Leader creates version
- **How to find a leader?**



source

Paxos History

- **Leslie Lamport discovered the Paxos algorithm in late 1980s**
  - Attempt to prove that there was no such algorithm which can tolerate the failure of any number of its processes
  - Until he realized that he created working protocol

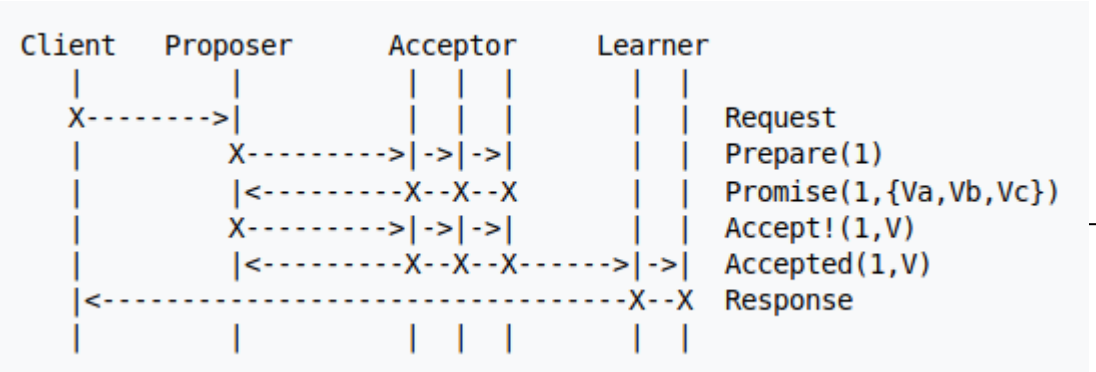


source

- **Wrote paper and submitted it to Transactions on Computer Systems (TOCS) in 1990**
  - Reviewer: was mildly interesting, but needs significant improvement
  - Leslie Lamport: “so I did nothing with the paper”
- **People started to using Paxos to solve problems in distributed systems**
- **Resubmitted in 1998 to TOCS**
  - Accepted without any major changes
- **Paxos paper won an ACM SIGOPS Hall of Fame Award in 2012**
- **Received Turing award in 2013, also due to Paxos**
  - “Turing Award is generally recognized as the highest distinction in computer science and the "Nobel Prize of computing“”

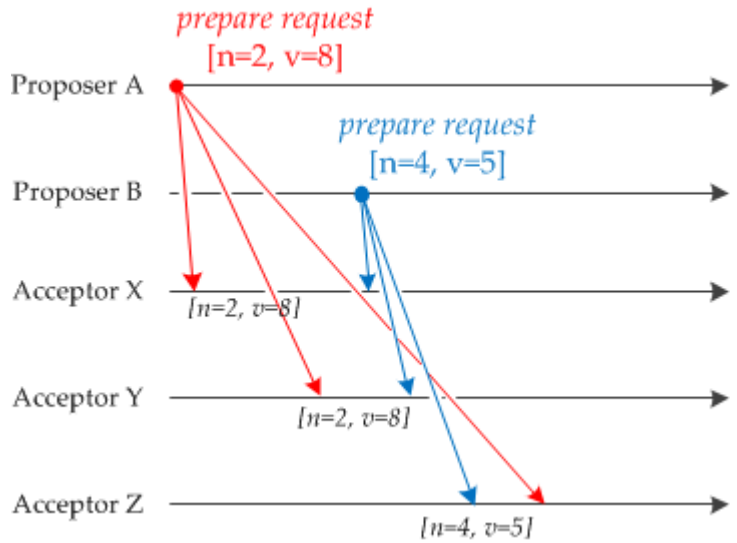
Paxos consensus

- **Paxos: considered difficult to understand**
  - "This website explains the infamously difficult to understand Paxos consensus protocol"
  - "Paxos, a really beautiful protocol for distributed consensus"
  - "Paxos is an algorithm whose entire behaviour is subtly difficult to grasp"
- **Problem: want reliable computing,**
  - But: have unreliable components
- **Due to unreliable components, run multiple components, i.e, multiple servers**
  - Replica: inconsistent state
- **Paxos guarantees that nodes will only ever choose a single value, but does not guarantee that a value will be chosen if a majority of nodes are unavailable**
- **Roles: proposer, acceptor, and learner**
  - Proposer proposes a value that it wants agreement upon
  - Acceptor gets proposal, makes promises, sends result to learners
  - Learners: majority of acceptors must choose the same value
- **2 phases: prepare/promise phase, accept phase**

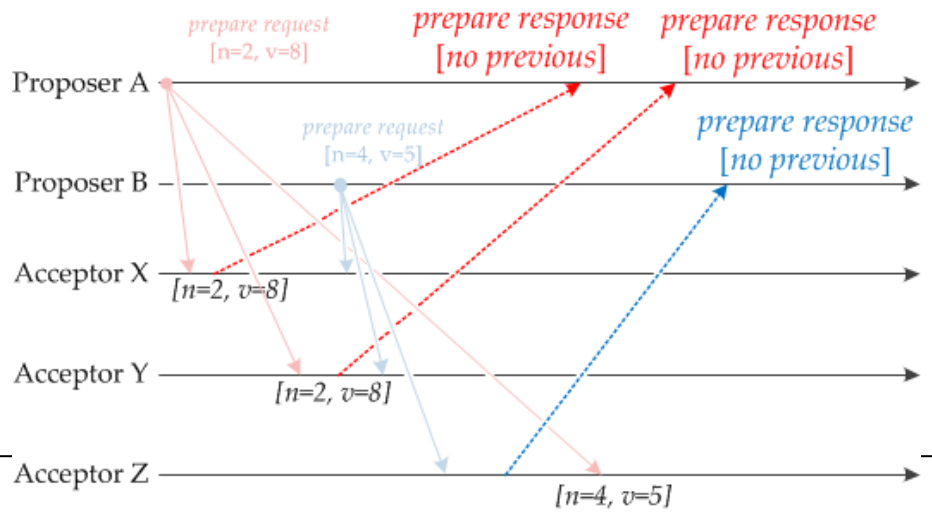


Paxos consensus

- **Proposer: prepare and accept requests, proposal number and value (n, v)**
  - Acceptor: already seen higher proposal number: ignore
  - Acceptor: seen lower proposal number: send back highest accepted n,v.

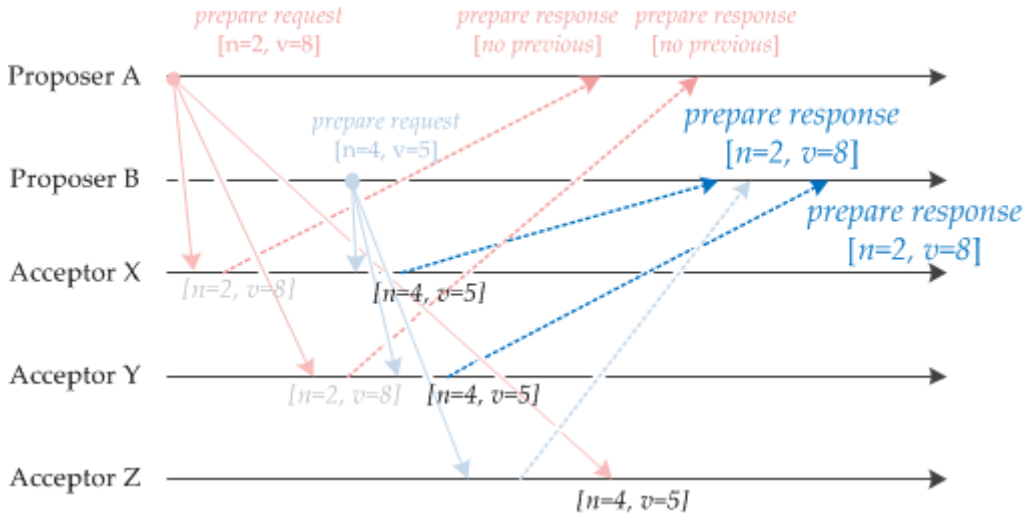


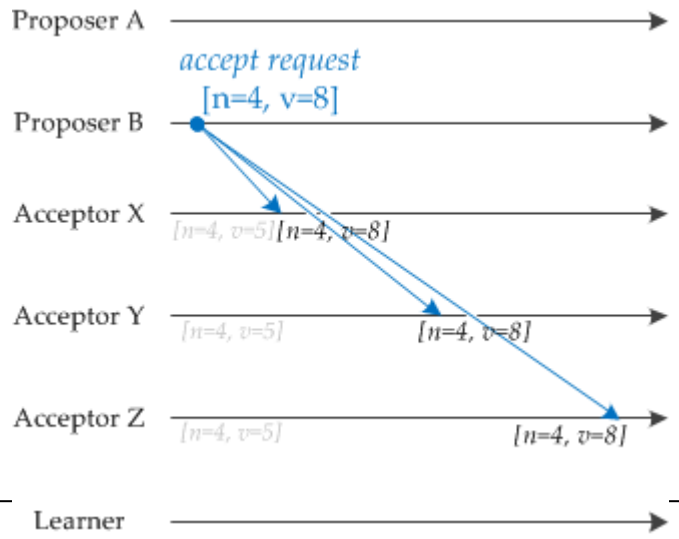
- **Proposer A and proposer B**
  - Acceptor Z receives B before A
- **If acceptor receiving a prepare request for the first time**
  - Acceptor responds with a prepare response
  - Promises never to accept another proposal with a lower proposal number



Paxos consensus

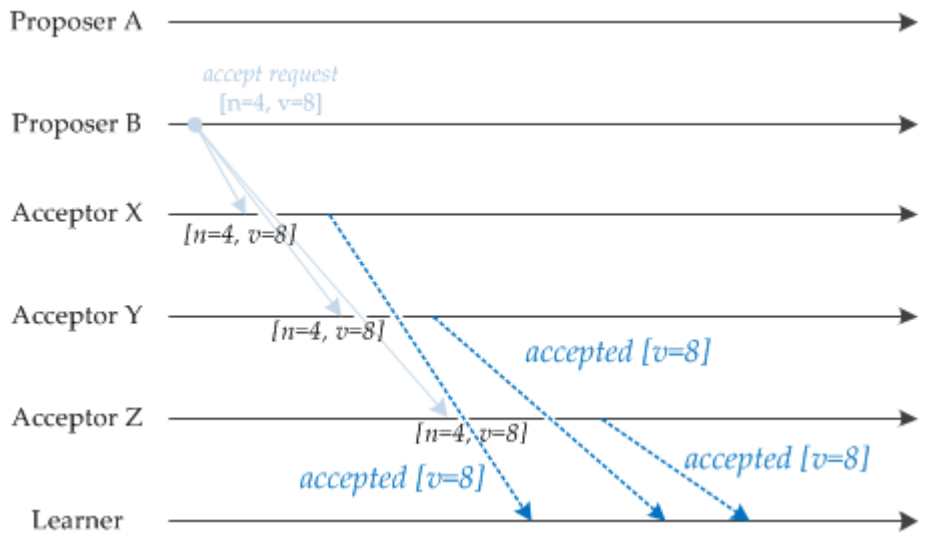
- **Acceptor Z receives proposer A's request, and acceptors X and Y receive proposer B's request.**
  - Only accept requests with higher number, Acceptor Z not sending response (or negative response) to B.
- **Proposer B sends an accept request to each acceptor containing the proposal number it previously used (n=4) and the value associated with the highest proposal number among the prepare response messages it received (v=8)**
- **Proposer A sends its accept request before proposer B, but acceptor ignores them**





Paxos consensus

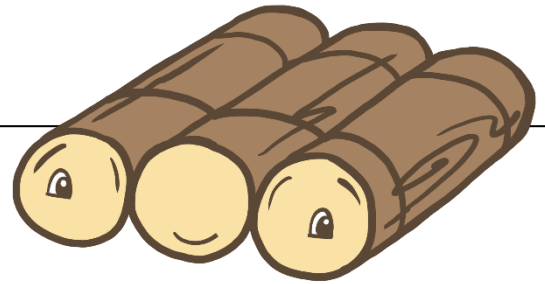
- **If an acceptor receives an accept request for a higher or equal proposal number than it has already seen, it accepts and sends a notification to every learner node.**
  - A value is chosen by the Paxos algorithm when a learner discovers that a majority of acceptors have accepted a value
  - Once a value is chosen by Paxos, communication with other proposers cannot change value
  - If another proposer, sends a higher proposal number than has previously been seen, with a different value (e.g., n=6, v=7), each acceptor responds with the previous highest proposal (n=4, v=8)
  - This requires the proposer to send an accept request containing [n=6, v=8], which confirms the value that has already been chosen



- **Leader election: the leader is the node that has its data chosen by the Paxos instance**

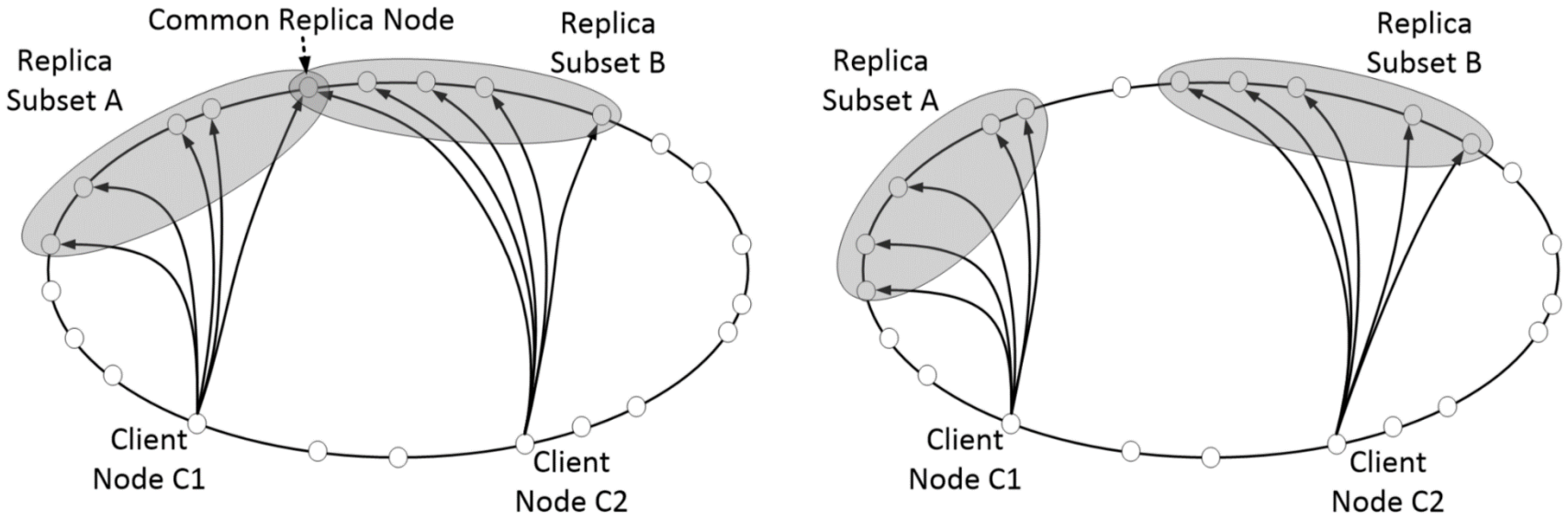
Raft (multi paxos)

- “this makes Raft more understandable than Paxos and also provides a better foundation for building practical systems. ”
- RAFT: **Reliable, Replicated, Redundant, And Fault-Tolerant**
- **Follower, Candidate, Leader**
  - Raft implements leadership election,
  - Once a leader has been elected, all decision-making within the protocol will then be driven only by the leader
  - Only one leader can exist at a single time
- **Each follower has a timeout (typically between 150 and 300 ms) in which it expects the heartbeat from the leader.**
  - The system is only available when a leader has been elected and is alive
  - Otherwise, a new leader will be elected and the system will remain unavailable for the duration of the vote
  - Starts election by increasing term counter, voting for itself, and sending a message to all other servers requesting their vote
  - If a higher term is received, become follower, if not, leader



Consistency

- **Consistency in DHTs – vDHT, similarities to Paxos**
  - Number = versions, for doing updates
  - Simplified roles (peer)
  - No leader election, works well with churn (not heavy churn)
- **CoW, software transactional memory (STM) → for consistent updates. Works for light churn**



Consistency

- **vDHT Basics**
  - No locking, no timestamps
  - Every update – new version
    - 1. get() latest version, check if all replica peers have latest version, if not wait and try again

      - may add delay

      + wait until update is completed
    - 2. put() prepared with data and short TTL, if status is OK on all replica peers, go ahead, otherwise, remove the data and go to step 1.
      - Data can be either send now or with the confirm, if sent now we are optimistic
      - Peer marks the value as prepared, other put() fail on that key. If nothing happens, TTL
      - Value linked to previous version(s) (hash)
    - 3. put() confirmed, don't send the data, just remove the prepared flag and reset TTL
- **In case of heavy churn, API user needs to resolve**
  - Get latest version may return fork
  - Abort or resolve (join) manually

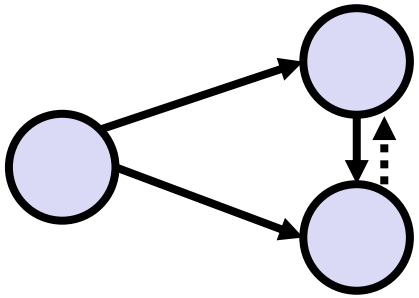
URL: b.socrative.com/student/

Room: HSR

Kademlia Broadcasting / Flooding in P2P

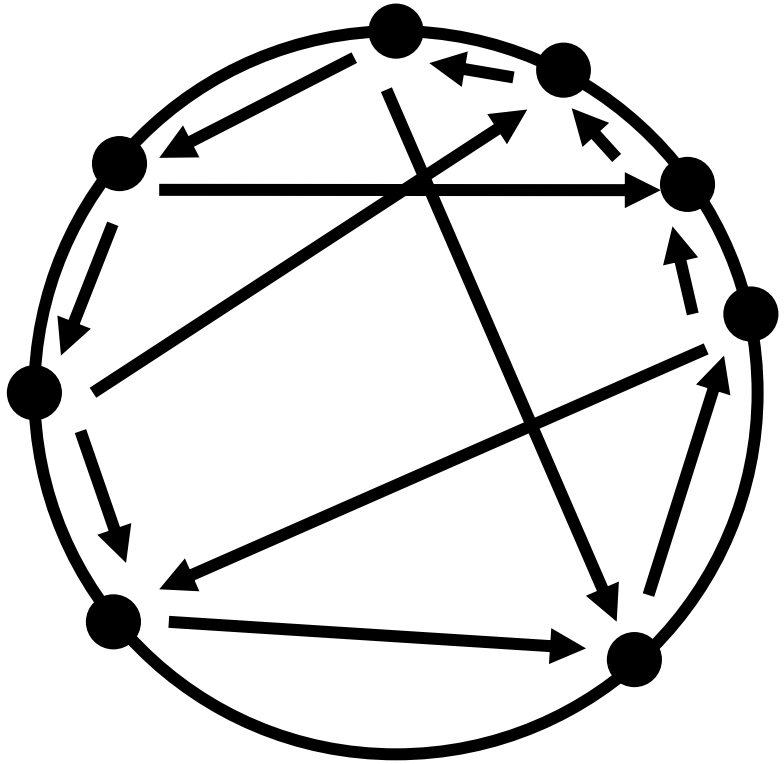
- **Flooding scales poorly**
  - Unnecessary traffic
  - 5 peers, fully meshed: 4 msg, 4 x 3 msg → 16 msg
- **Reduce Messages**
  - Don’t send to all neighbors (random walk)
  - Coupon collector's problem
    - Number of trials grow O(n log n),
    - Reach 50 peers, it takes about 225 trials to reach all 50 peers
    - Tradeoff: all peers vs. most peers / many msgs vs. reduce msgs
- **Broadcasting use-cases**
  - Broadcast global parameters, global events (maintenance)
  - Hierarchical system: broadcast root inode (e.g. / in a file system)
  - Send message to all players (challenge task)

- **Ethereum Wispher: TTL, “probabilistic message forwarding”**



Broadcasting in Structured Systems

- **5 peers fully meshed: How to send only 4 messages?**
- **Structured vs. unstructured:**
  - Knowing the direction
- **Broadcasting in ring-based Overlay**
  - Main idea: tell other peers in what direction so send
- **Example:**
  - Send to neighbor (successor)
  - For faster spreading / redundancy send to far away peer (overhead)
  - Overhead: 4 messages
- **Kademlia? Tree-based structure**



Broadcasting in Kademlia

- **Tree structure – no direction, example 7 peers**
- **Main idea: send to random peer in bag X**
  - Send along in which bag this peer was, only send to smaller bags

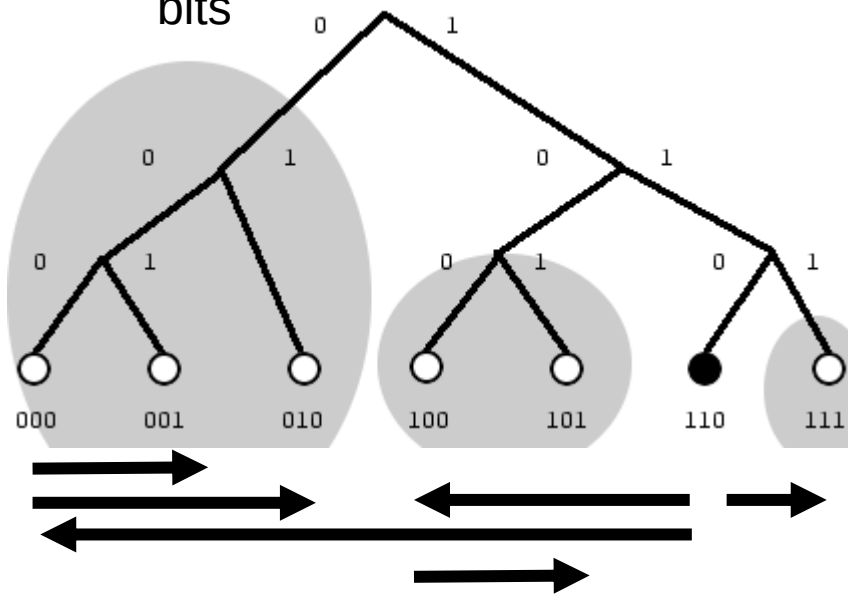
Peer 6 (110) broadcast to b1,b2,b3		
7 (b1)	4 (or 5) (b2)	0 (or 1, 2) (b3)

Peer 4 (100) broadcast to b1		
5 (b1)	- 6	- 0

Peer 0 (000) broadcast to b1,b2		
1 (b1)	2 (b2)	- 6

- **Peer 4 and 0 send simultaneously**

Routing with XOR, with 3-bits



Broadcasting in Kademlia

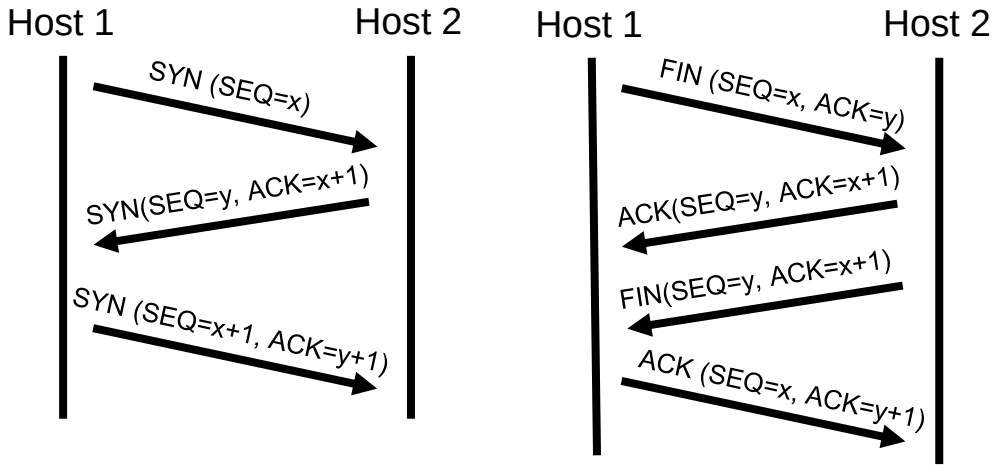
- **7 peers, 6 messages**
  - Works only with perfect routing, churn?
- **Redundancy: send to R random peer in bag X**
  - Previous example: R=1, TomP2P: R=3
- **Demo: StructuredBroadcastHandler**
  - 1000 peers
  - FROM_EACH_BAG = 3, ~3800 messages
  - No redundancy: 999 messages
- **Testcase/showcase:**
  - net.tomp2p.p2p.TestBroadcast.testBroadcastTCP()

URL: b.socrative.com/student/

Room: HSR

Protocol, Serialization, VSS recap: Layer 4 - TCP

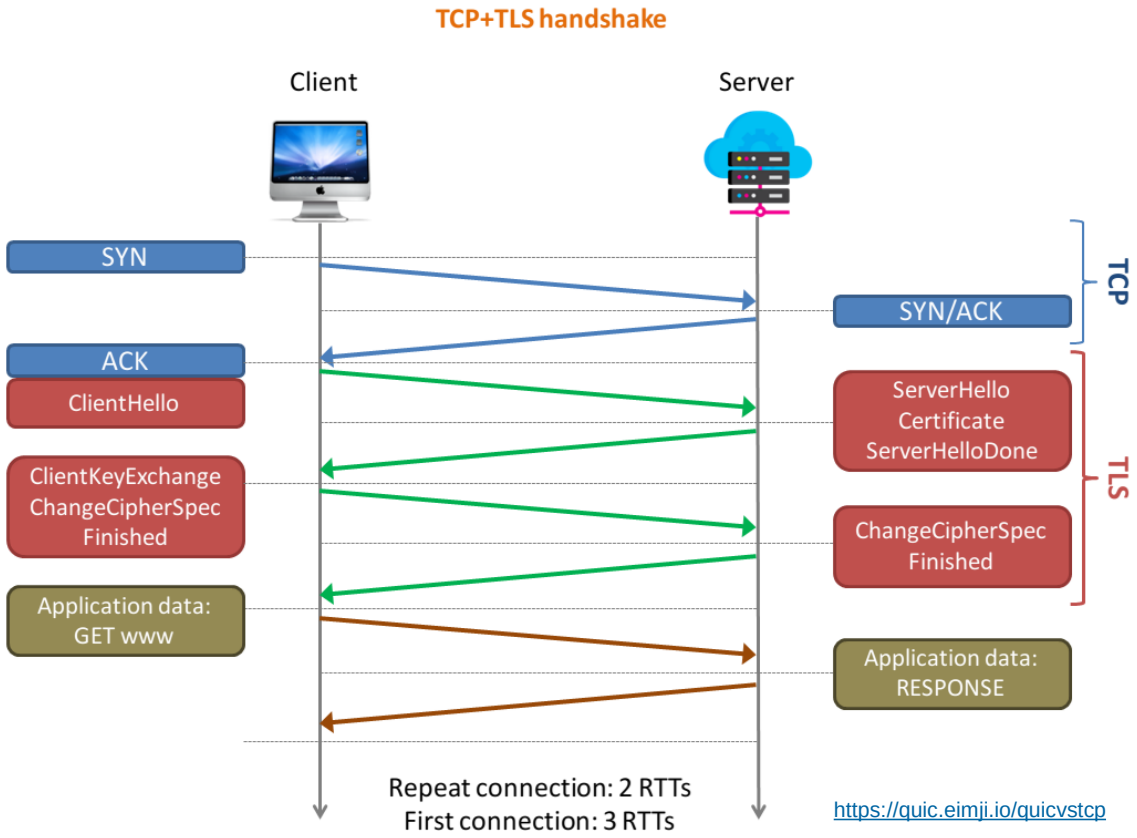
- **Connection establishment**
  - SYN, SYN-ACK, ACK (three way)
  - Initiates TCP session: initial sequence number is ~random
  - Congestion control
- **Connection termination**
  - FIN, ACK + FIN, ACK (three/four way)
  - 3-way handshake, when host 1 sends a FIN and host 2 replies with a FIN & ACK
- **Sequences and ACKs**
  - Identification each byte of data
  - Order of the bytes → reconstruction
  - Detecting lost data: RTO, DupACK:



- **Retransmission timeout**
  - If no ACK is received aftert timout (e.g. 2xRTT), resend.
- **Duplicate cumulative acknowledgements**
  - ACKs for last consecutive packets
  - 3 times same ACK → retransmit (fast retransmit)

Protocol, Serialization

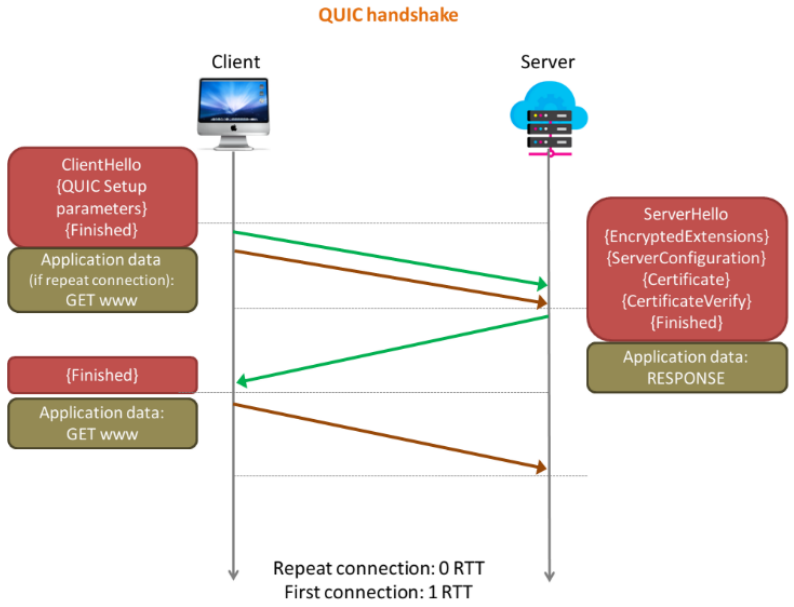
- **Security: Transport Layer Security (TLS)**
  1. "client hello" lists cryptographic information, TLS version, ciphers/keys
     - Forward secrecy: ephemeral keys
  2. "server hello" chosen cipher, the session ID, random bytes, digital certificate (checked by client), optional: "client certificate request"
  3. Key exchange using random bytes, now server and client can calc secret key
  4. "finished" message, encrypted with the secret key



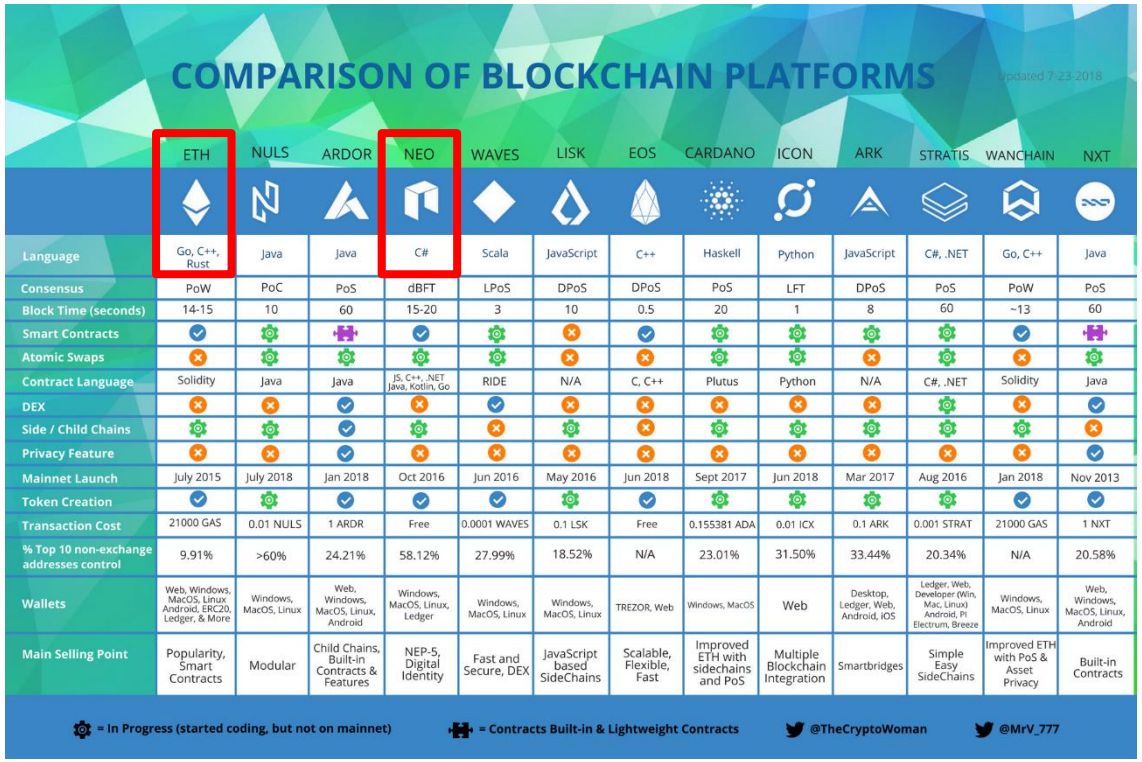
https://www.ibm.com/support/knowledgecenter/SSFKSJ_7.1.0/com.ibm.mq.doc/sy10660_.htm

Protocol considerations

- **Ping to Australia: 329ms**
  - One way ~ 165ms
- **TCP + TLS handshake:**
  - 3RTT = 987ms! No data sent yet
- **QUIC: 1RTT, TLS 1.3: 2RTT**



- **Block time Bitcoin 10min**
- **Blockchains today 60s – 0.5s (tomo)**



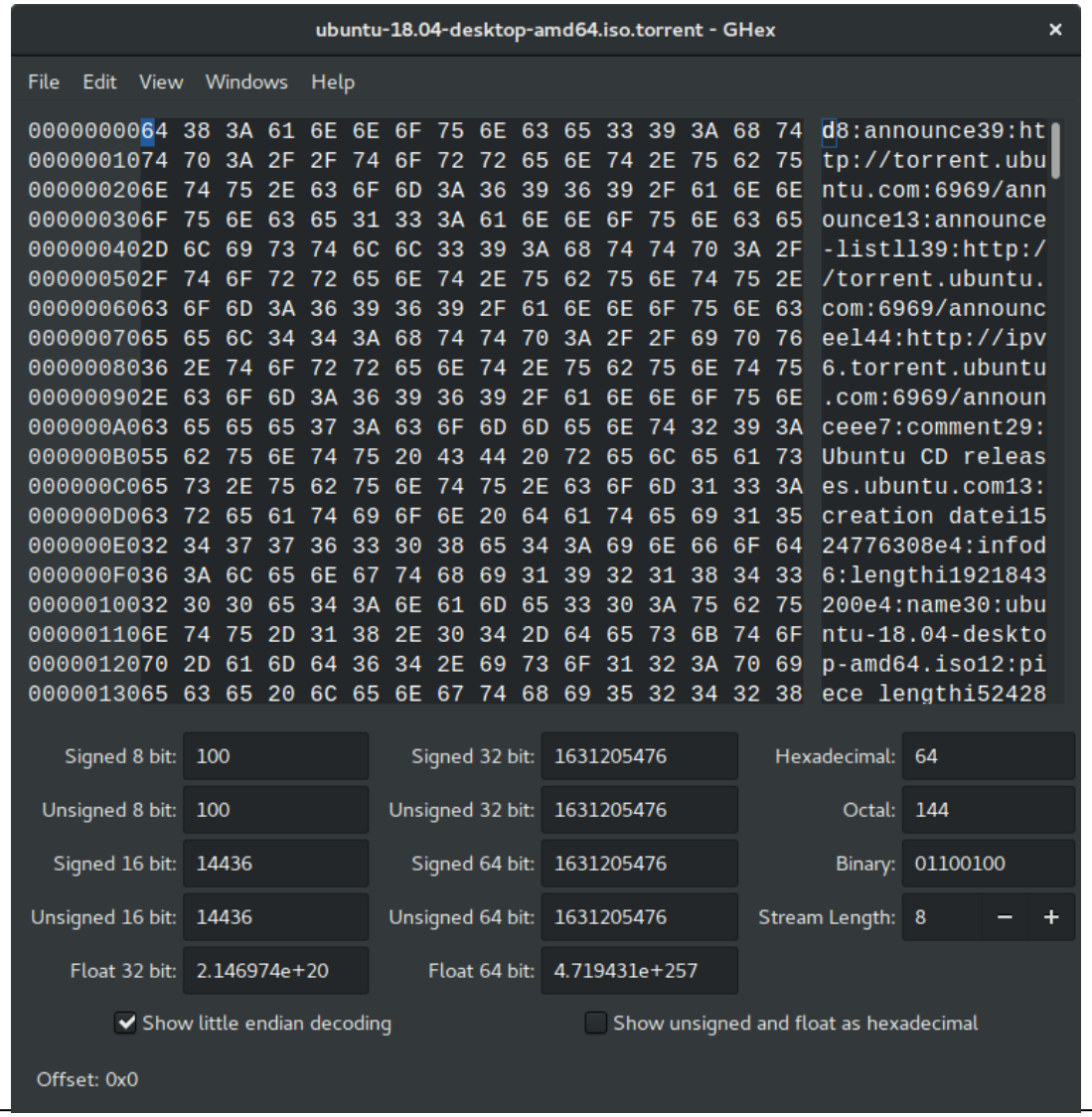
https://twitter.com/thecryptowoman/status/1021385599088054273

URL: b.socrative.com/student/

Room: HSR

Protocol, Serialization, VSS recap: Protocols

- **Avro / gRPC / ASN1**
- **JSON encoding, e.g., JSON-RPC**
- **Benconding**
  - Integers: i42e
  - Byte string: 4:test
  - List: l4:testi42ee
  - Map/dictionary: d4:test3:hsr3:tomi42ee



Serialization in Ethereum

- **RLP (Recursive Length Prefix): encode arbitrarily nested arrays of binary data**
  - Integers represented in big endian
  - String 0x8[len]: [0x83, 'd', 'o', 'g']
    - Data is string if in range [0x80, 0xb7]
    - Larger data, 0xb8, next byte is len
    - 0xb9, next 2 bytes len, … 0xbf 8 bytes len
    - At most 64bit length
  - Single byte below 55: 15 = 0x0f
  - List 0xc[len]: [0xc8, 0x83, 'c', 'a', 't', 0x83, 'd', 'o', 'g']
- **Parameter encoding:**

```
contract Foo {
  function bar(bytes3[2] memory) public pure {}
  function baz(uint32 x, bool y) public pure returns (bool r) { r = x > 32 || y; }
  function sam(bytes memory, bool, uint[] memory) public pure {}
}
```

- **Method ID: first 4 bytes of hash of method signature**
  - baz(uint32,bool) → 0xcdcd77c0
- **`web3.eth.abi.encodeFunctionSignature('myMethod(string,uint256)')`**
- **`new FunctionBuilder("myMethod").addInput("string", "hallo").addInput("uint256", new BigInteger("10000"))`**
  - Uint32, e.g. 45: 0x0000000000000000000000000000000000000000000000000000000000000045
  - Boolean, e.g. true: 0x0000000000000000000000000000000000000000000000000000000000000001
  - Onchain: 0xcdcd77c0000000000000000000000000000000000000000000000000000000000000004500000000000000000000000000000000000000000000000000000000000000001
  - Efficiency?

Serialization in Ethereum

- **Dynamic types**
  - `sam(bytes,bool,uint256[])`
  - Location of the data (offset):
    `0x0000000000000000000000000000000000000000000000000000000000000060`
  - `Boolean:`
    `0x0000000000000000000000000000000000000000000000000000000000000001`
  - `Locatoin of data (offset):`
    `0x00000000000000000000000000000000000000000000000000000000000000a0`
  - `At position 60, length of byte array:`
    `0x0000000000000000000000000000000000000000000000000000000000000004`
  - `“dave” (left padded):`
    `0x6461766500000000000000000000000000000000000000000000000000000000`

- **Number of items in array at position 0xa0**
  - 0x0000000000000000000000000000000000000000000000000000000000000003
  - 3 times data item:
    - 0x0000000000000000000000000000000000000000000000000000000000000001
    - 0x0000000000000000000000000000000000000000000000000000000000000002
    - 0x0000000000000000000000000000000000000000000000000000000000000003

- **In total 292 bytes, 16 bytes non-zero**
  - Efficiency? Example

Questions?

Dr. Thomas Bocek thomas.bocek@hsr.ch