Byzantine General Problem

A mini-lecture series

CSE498 Collaborative Design (W) - Secure and Efficient C++ Software Development

04/14/2025

Kira Chan

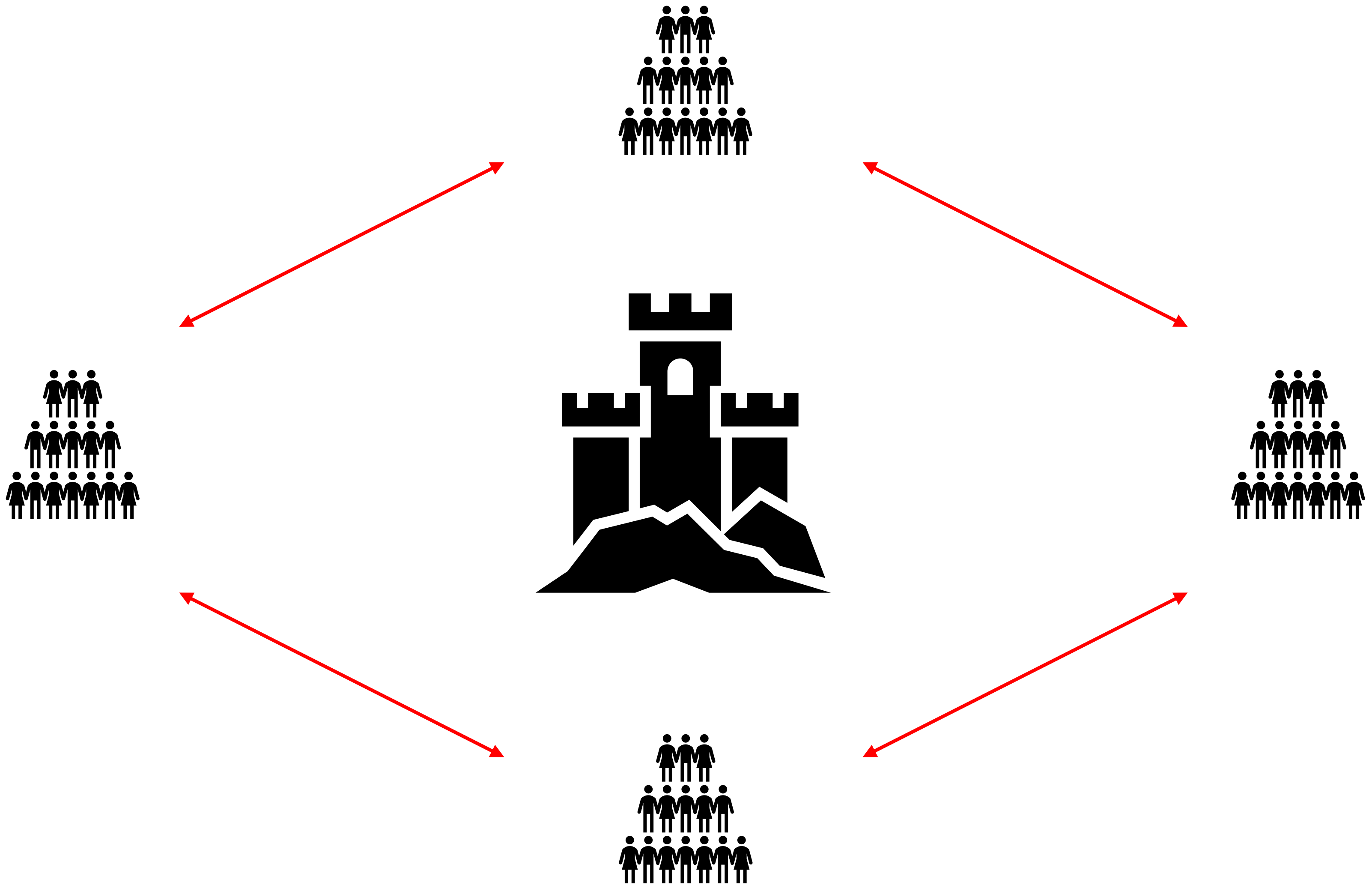
https://cse.msu.edu/~chanken1/

Problem

- In distributed computing, we have a notion of ***consensus***
- ***Consensus:*** how do we get multiple processes to agree

- Byzantine general problem demonstrates an instance of such problem
  - Created by Leslie Lamport et al. as an analogy
  - Generalisation of the two general problem

Byzantine General Problem



Byzantine General Problem

- You are the general of one of the *n* armies surrounding the city
- You have an option to attack or retreat
- If all the armies attack at the same time, then the attack is successful
- If all the armies retreat, then it’s OK but you did not conquer the city
- If some of the armies attack while some retreat, then you will fail

Complications

- What if a messenger gets captured before your message reaches its recipient?
- What if the acknowledgement message gets captured?
- What if the messenger is captured and the message is replaced to do the opposite?
- What if one of the general is a traitor and purposely sabotages your attack plans?

Unsolvability

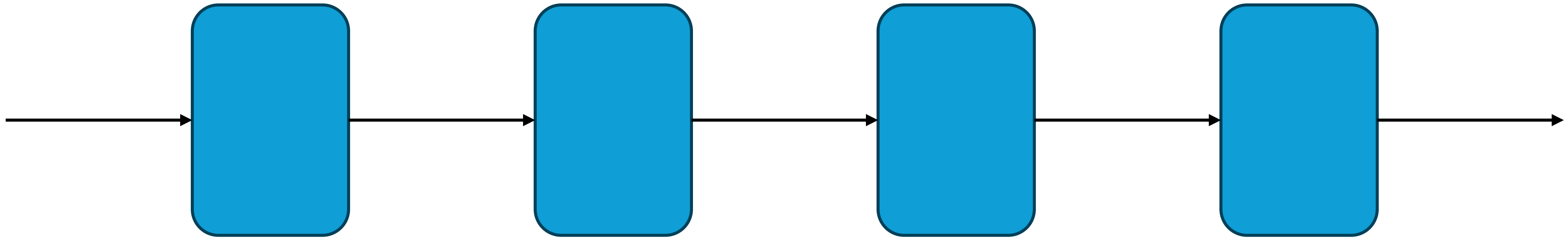
- In the general sense, this problem is unsolvable
- But it can be solved if we add some restrictions
  - If 2/3 of the generals are loyal

Byzantine Fault

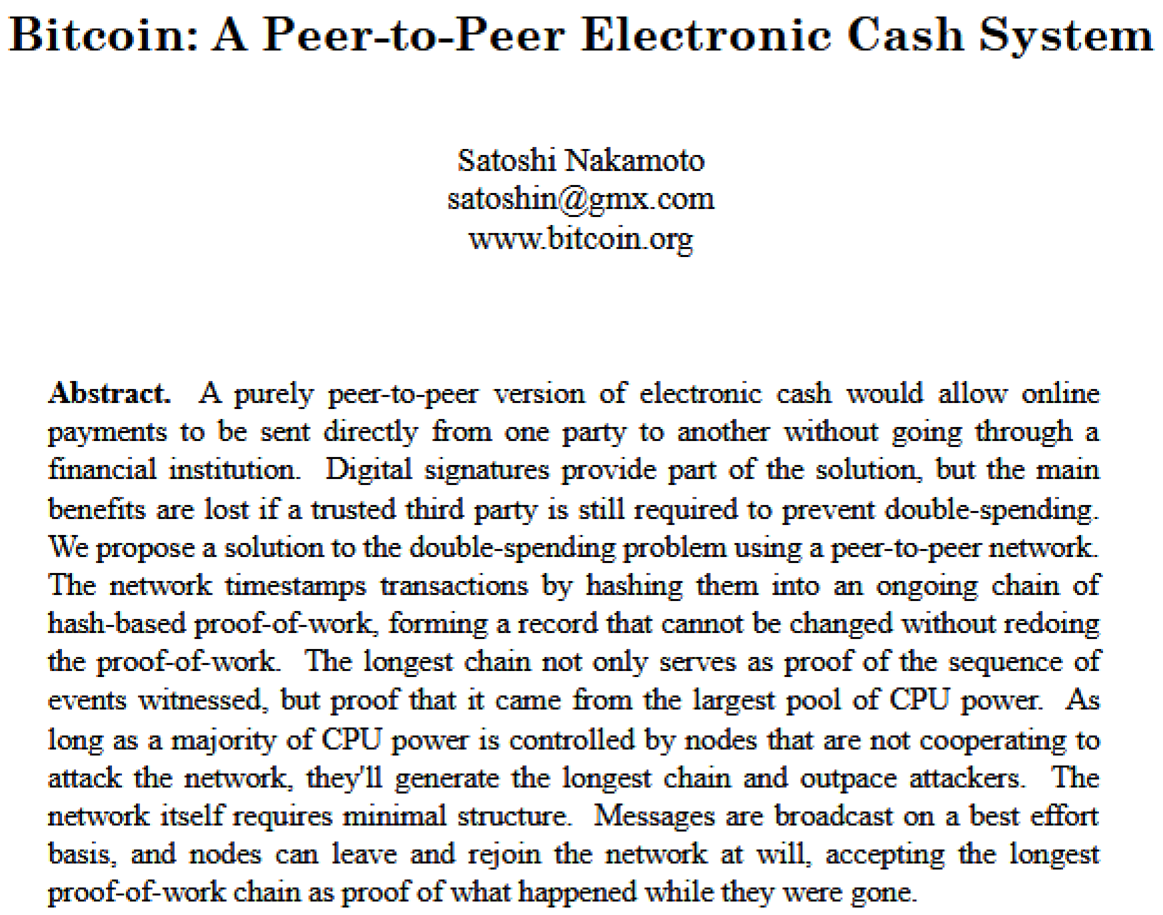
- Byzantine fault is a fault that presents different symptoms to different observers
  - This is usually caused by imperfect information
- **Byzantine fault tolerance** describes the resilience of your system against Byzantine faults
  - What are some systems that is fault tolerant?

Partial solution: blockchains

- Blockchain deals with a system where the is no centralised authority
- Unlike traditional databases (e.g., a bank), blockchains maintain a distributed database called the ***ledger***
- Blockchain removes a single point of failure and ensures that data is not easily modifiable



On the mystery of bitcoin



Bitcoin: A Peer-to-Peer Electronic Cash System

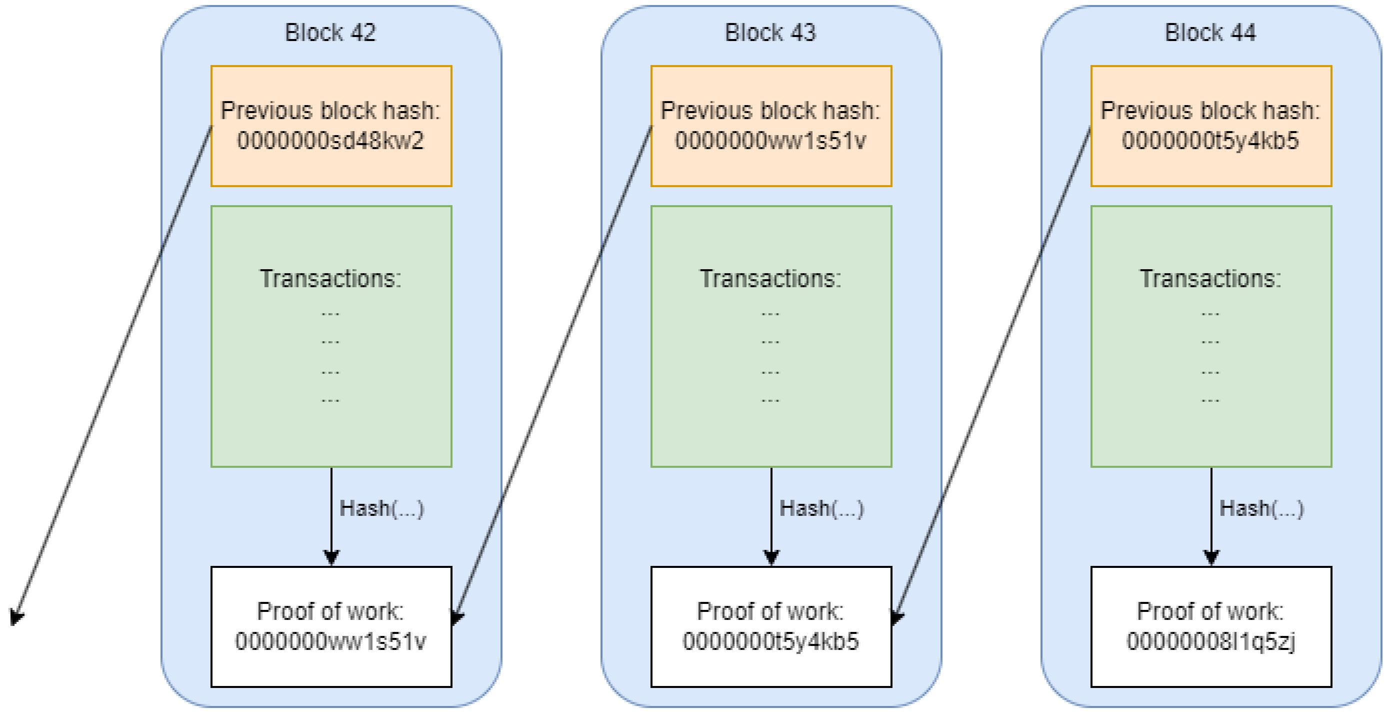
Satoshi Nakamoto
satoshin@gmx.com
www.bitcoin.org

Abstract. A purely peer-to-peer version of electronic cash would allow online payments to be sent directly from one party to another without going through a financial institution. Digital signatures provide part of the solution, but the main benefits are lost if a trusted third party is still required to prevent double-spending. We propose a solution to the double-spending problem using a peer-to-peer network. The network timestamps transactions by hashing them into an ongoing chain of hash-based proof-of-work, forming a record that cannot be changed without redoing the proof-of-work. The longest chain not only serves as proof of the sequence of events witnessed, but proof that it came from the largest pool of CPU power. As long as a majority of CPU power is controlled by nodes that are not cooperating to attack the network, they'll generate the longest chain and outpace attackers. The network itself requires minimal structure. Messages are broadcast on a best effort basis, and nodes can leave and rejoin the network at will, accepting the longest proof-of-work chain as proof of what happened while they were gone.

Blockchains

- The blockchain acts like a linked list
- Each node contains the following element:
  - 1) Hash of the previous block
  - 2) Transactions: monetary transactions or information to be stored
  - 3) Proof of work: a cryptographic puzzle
- Using blockchain, we remove the need to trust unknown nodes, rather trusting the collective system of nodes (which is assumed to be generally good)

Example blocks


Block 42
Previous block hash: 0000000sd48kw2
Transactions: ... ...
Hash(...)
Proof of work: 0000000ww1s51v
Block 43
Previous block hash: 0000000ww1s51v
Transactions: ... ...
Hash(...)
Proof of work: 0000000t5y4kb5
Block 44
Previous block hash: 0000000t5y4kb5
Transactions: ... ...
Hash(...)
Proof of work: 00000008l1q5zj

The creativity of blockchain: Proof of Work

- The proof of work is a piece of data that is difficult to produce yet easy to verify
- Recall what a hash function is

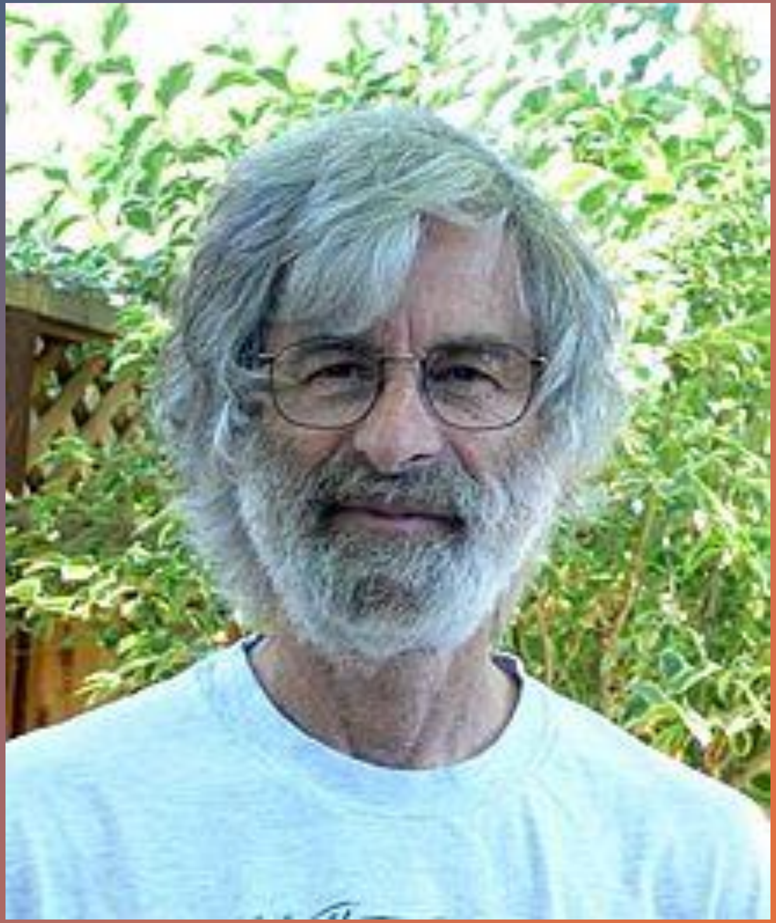
  - **Computationally hard to produce:** requires a node to correctly guess a “nonce” that when combined with the content of the block, produces a hash that starts with a given number of leading zeroes
  - **Computationally easy to verify:** verification is as simple as generating the node and ensuring it has the satisfying number of zeroes

Why it works

- The idea is blocks cannot be generated easily, and as such requires some amount of compute power to “sign” the blocks
- If an attacker wants to:
  - Modify a transaction (say add a digit to the amount transferred):
    - The new hash of the block will not have correct number of leading zeroes
    - They must recompute the hash to match
    - Increasingly harder as the blockchain gets longer
  - Add a malicious block or transaction:
    - Invalid transactions are rejected by other nodes (voting)
    - Including a block means the attacker is trying to outcompute all other nodes in the system
    - Since the longest block is used, they will have to keep outcomputing even if they successfully find a correct hash once

Summary

- From the original blockchain paper
  - The longest blockchain serves as a proof of the sequence of events witnessed, but also that it comes from the largest pool of computing power
  - Records are immutable
  - As long as majourity of the CPU power is controlled by nodes not trying to attack the system (they can each mind their own thing), they will generate the longest chain and outpace the attacker
- Applications are beyond currency, can be used to verify software, history records, voting records, etc.



Person of the Day Leslie Lamport

- 2013 Turing award winner
- Initial developer of LaTeX
- Distributed systems (Byzantine general problem)
- Lamport’s clock
- Temporal Logic
- Lamport’s bakery algorithm