NAKAMOTO CONSENSUS

Kai Mast
CS639/839
Spring 2023

ANNOUCEMENT: UPCOMING TALKS

Natacha Crooks (UC Berkeley)

“Basil, a new transactional Byzantine Fault Tolerant key-value store”

- Friday (2/24) at noon
- Computer Sciences 2310 (limited space)

Lorenzo Alvisi (Cornell)

“Orderrr! A tale of Money, Intrigue, and Specifications”

- Monday (2/27) at noon
- Via Zoom

AGENDA

Basics

- Recap: Peer-to-Peer Networks
- Gossip Protocols
- What is Consensus?

Break 1

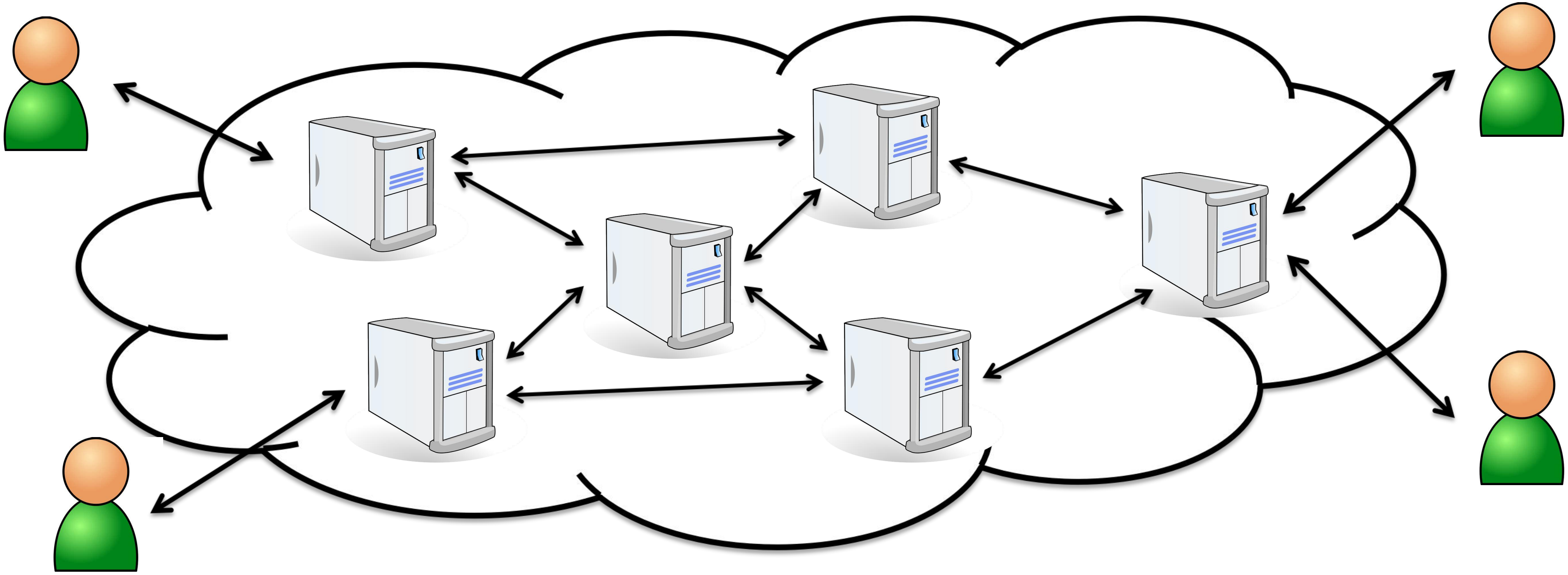
Nakamoto Consensus

Break 2

More Details

- Block Size and Block Frequency
- GHOST

RECAP: PEER-TO-PEER NETWORKS



- Not all nodes are known to everybody else
- No fixed topology
- Each node is only connected to a few peers

MESSAGE PROPAGATION

How does a message, e.g., a transaction, reach all nodes in a network?

Naive Solution: Directly send the message to every node in the network

Why does that not work?

- There are many (hundreds) of nodes
- Not all nodes may be known
    - We cannot directly talk to them

More Challenges

- Need to tolerate network failures
- Nodes might join or leave at any time

GOSSIP PROTOCOLS

Idea: Use the peer-to-peer network to *disseminate* message across the network

Approach:

- When you create a message, send it to all your peers
- When you receive a message, send it to all your peers

Very Scalable:
Each node only needs to forward message to a constant number of peers, independent of the network size

Failure resilient:
If individual nodes or networks links failed, message will spread through a different path

GOSSIP: IMPLEMENATION ATTEMPT

```
def create_message(self, content):
    msg = {
        “uid”: random(),
        “content”: content,
    }

    for peer in self.peers:
        peer.send(msg)
```

```
def on_receive(self, msg):
    for peer in self.peers:
        peer.send(msg)
```

Problem: Can create infinite loops

GOSSIP: IMPLEMENATION

```
def create_message(self, content):
    msg = {
        “uid”: random(),
        “content”: content,
    }

    # reuse code
    self.on_receive(msg)
```

```
def on_receive(self, msg):
    if msg[“uid”] in self.known_messages:
        # ignore duplicates
        return

    for peer in self.peers:
        peer.send(msg)

    self.known_messages.insert(msg[“uid”])
```

More Possible Optimizations:

- Don’t send message back to the peer we received it from
- Only advertise a message, and send it if needed

GOSSIP IN BLOCKCHAINS

```
def create_message(self, content):
    msg = {
        “uid”: random(),
        “content”: content,
    }

    # reuse code
    self.on_receive(msg)
```

```
def on_receive(self, msg):
    if msg[“uid”] in self.known_messages:
        # ignore duplicates
        return

    if not is_valid(msg.content):
        # ignore invalid blocks/txns
        return

    for peer in self.peers:
        peer.send(msg)

    self.known_messages.insert(msg[“uid”])
```

GOSSIP IN BITCOIN

Most Important Message Types

`inv` ("Inventory"): Advertise new blocks/transactions to your peers

`getdata`: Request specific blocks/transactions from a peer

`block` or `tx` ("transaction"): Send the requested data

Full Bitcoin Protocol Specification:
https://en.bitcoin.it/wiki/Protocol_documentation

CONSENSUS

Make multiple nodes agree on the same thing, e.g., which transaction to accept.

Why is this hard?

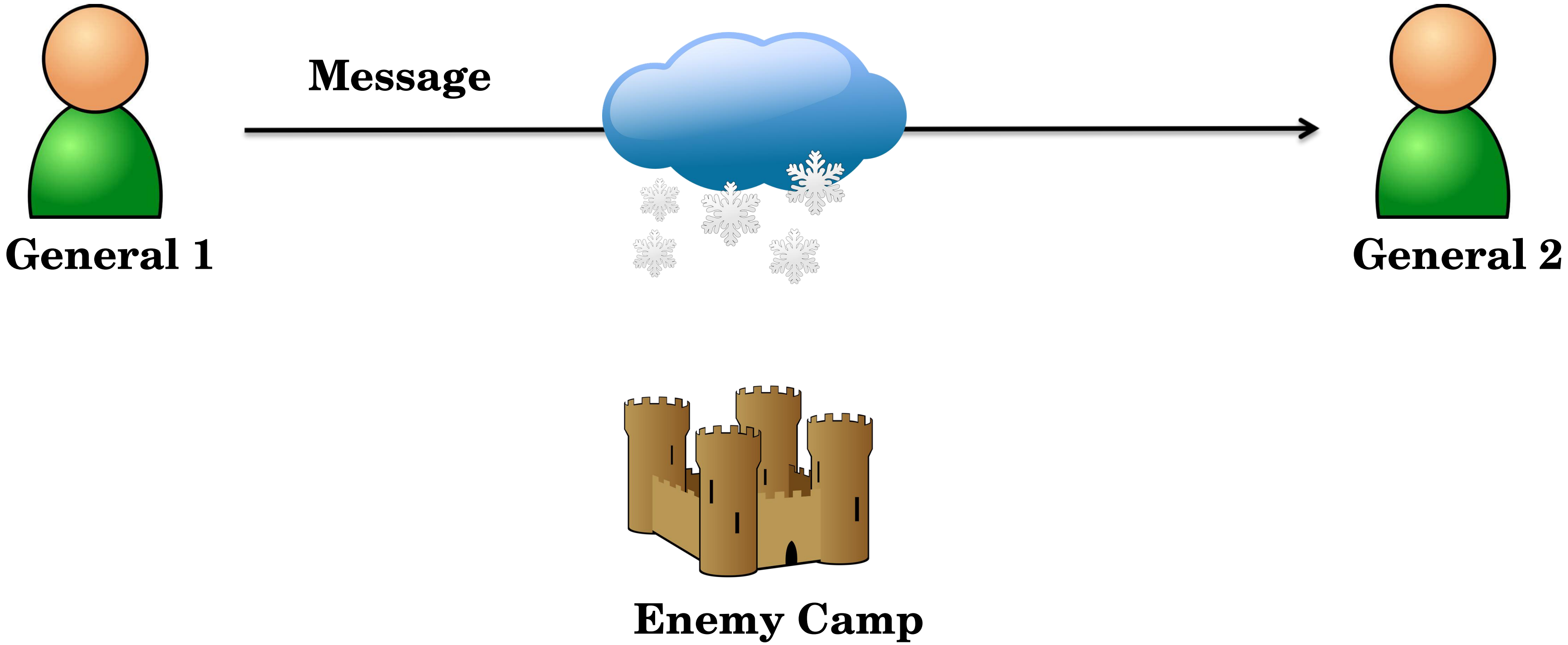
- Nodes can be faulty or malicious*
- Network delays might drop or reorder messages

Why is this even harder in the public blockchain setting?

- Large scale network
- (Usually) not all participants are known
- Nodes have different hardware and operating systems

*malicious behavior is technically also a type of failure

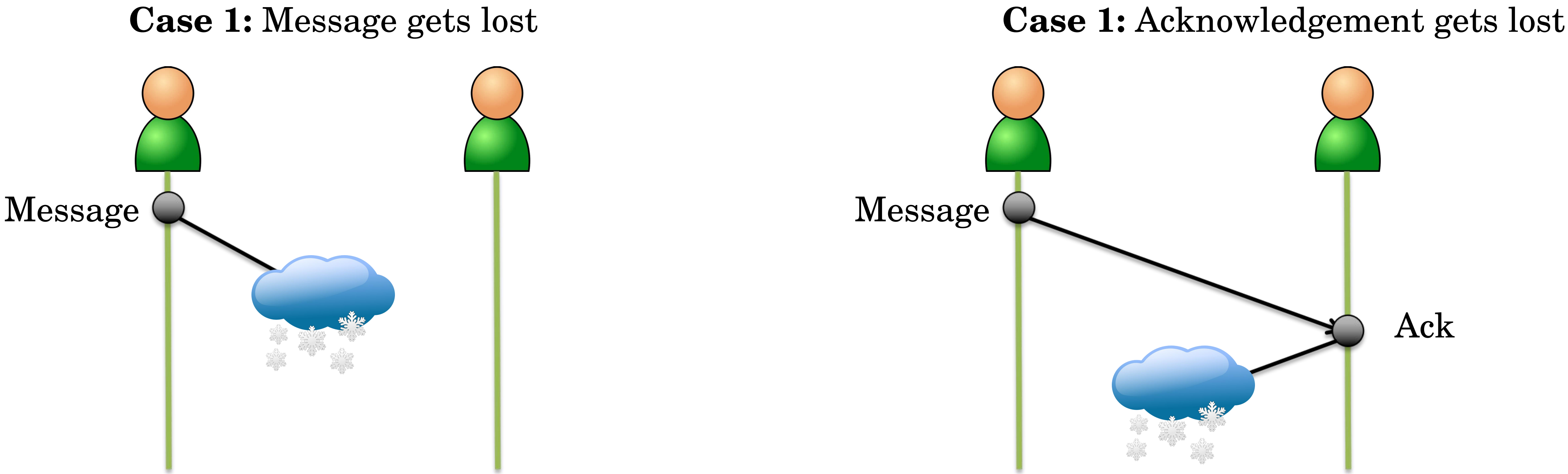
THE TWO (WISCONSIN) GENERALS' PROBLEM



Example to illustrate the challenges in achieving consensus

- Two generals have agreed to attack an enemy's camp
- The attack only succeeds if they attack at the same time
- They need to send a messenger through a snowstorm to agree on a time
  - There is a possibility the messenger dies in the storm and the message is lost

THE TWO (WISCONSIN) GENERALS' PROBLEM

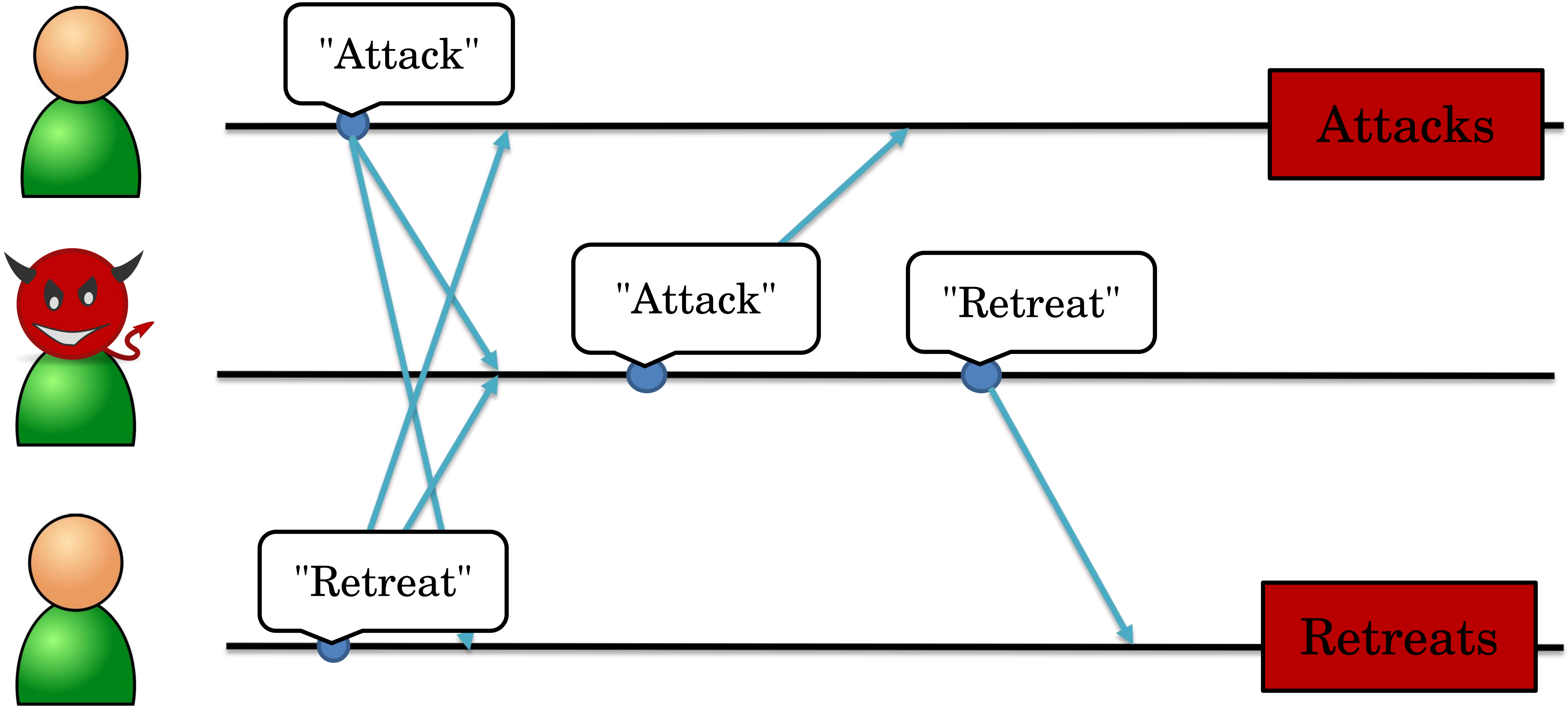


- To reach agreement, we need to know whether a message has arrived
- But the acknowledgement can also get lost

The problem, in this form, is unsolvable:

- We need *additional assumptions* about communication reliability to solve this

THE BYZANTINE GENERALS PROBLEM



- Generals need to agree on whether or not to attack
- A malicious minority can cause the honest generals to adopt a bad decision

BREAK 1

NAKAMOTO CONSENSUS

- Introduced with the Bitcoin paper in 2009
  - Named after the inventor's pseudonym

A new class of consensus protocols

- First permissionless/public protocol
- Behaves in a probabilistic fashion
- Works well with large-scale networks

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.

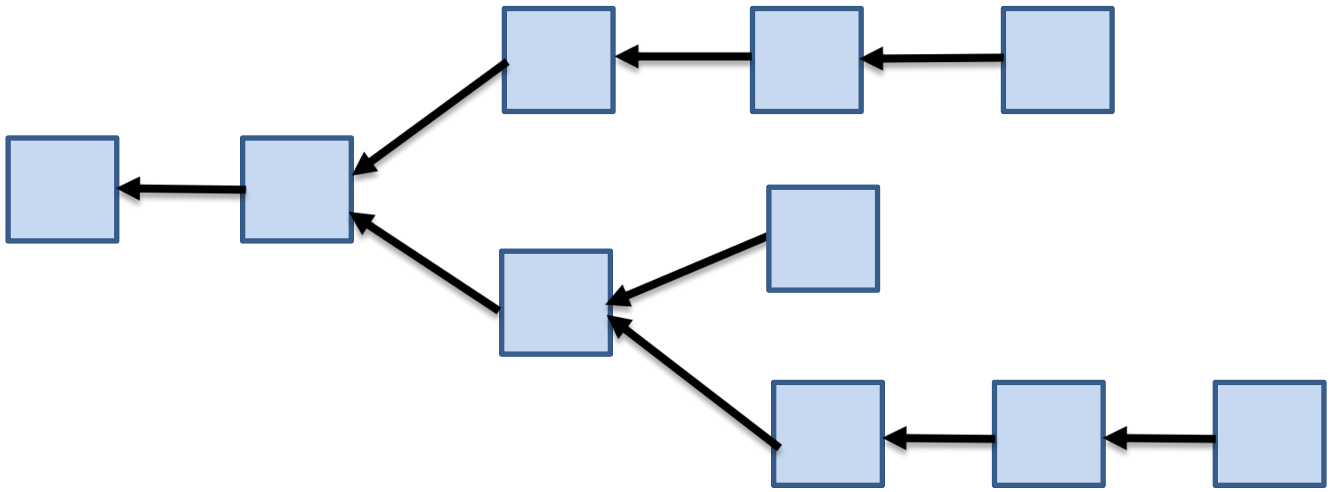
1. Introduction

Commerce on the Internet has come to rely almost exclusively on financial institutions serving as trusted third parties to process electronic payments. While the system works well enough for most transactions, it still suffers from the inherent weaknesses of the trust based model. Completely non-reversible transactions are not really possible, since financial institutions cannot avoid mediating disputes. The cost of mediation increases transaction costs, limiting the minimum practical transaction size and cutting off the possibility for small casual transactions, and there is a broader cost in the loss of ability to make non-reversible payments for non-reversible services. With the possibility of reversal, the need for trust spreads. Merchants must be wary of their customers, hassling them for more information than they would otherwise need. A certain percentage of fraud is accepted as unavoidable. These costs and payment uncertainties can be avoided in person by using physical currency, but no mechanism exists to make payments

ASIDE: CONVENTIONAL CONSENSUS PROTOCOLS

- Existing protocols are *deterministic*
  - We know which blocks/transactions have been accepted with full certainty
- Existing protocols are permissioned/*private*
  - All involved nodes are known
  - Adding or removing a node requires reconfiguration
- Most protocols, e.g., Paxos or PBFT, rely on a *leader*
  - One node is elected and in charge of generating blocks
  - Detecting leader failure and electing a new leader is tricky

THE BLOCKCHAIN



Purpose 1: Store transaction data and determine transaction ordering

Purpose 2: Track agreement on which transactions are accepted

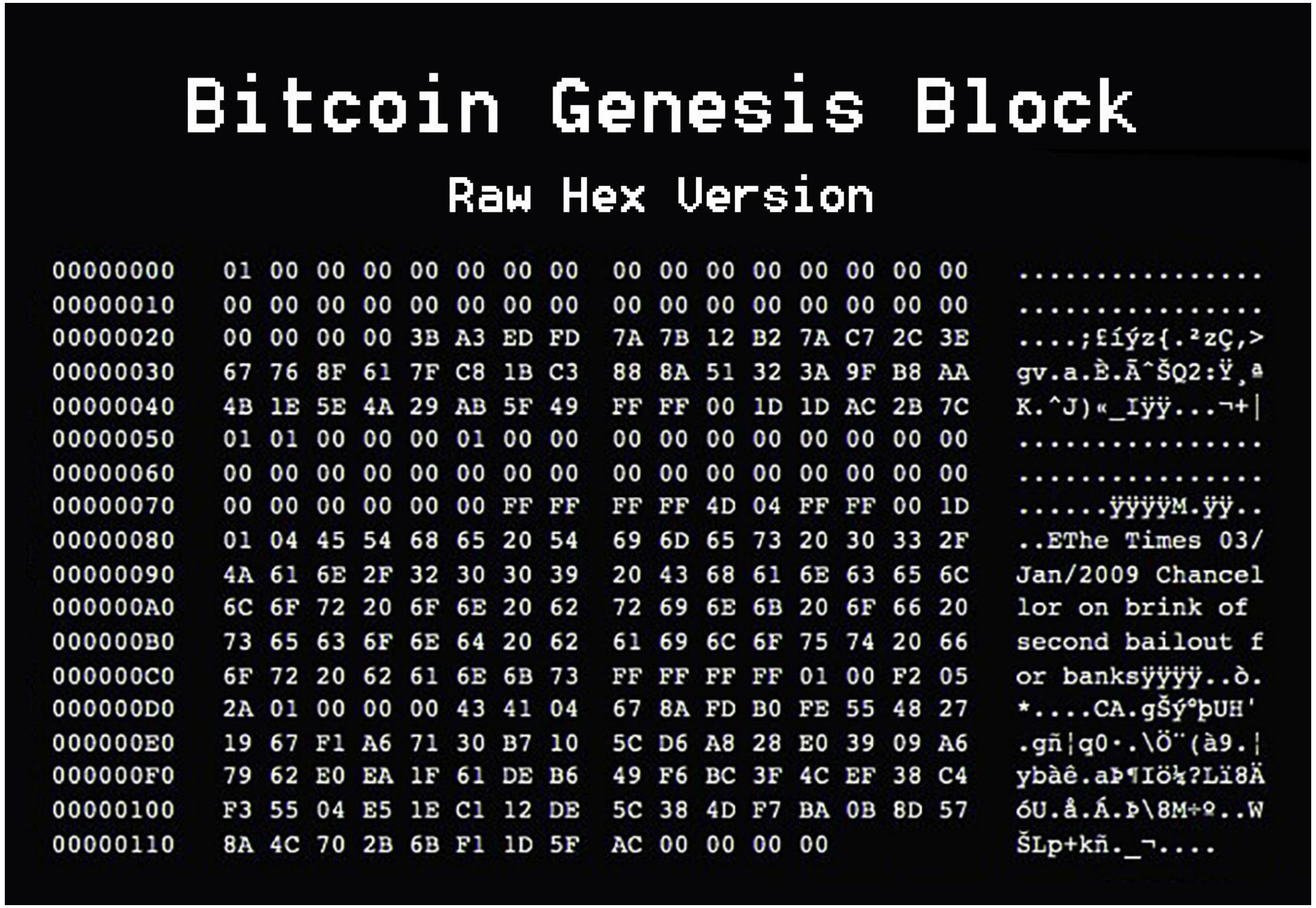
GENESIS BLOCK

There exists a single block as the "root" of every blockchain

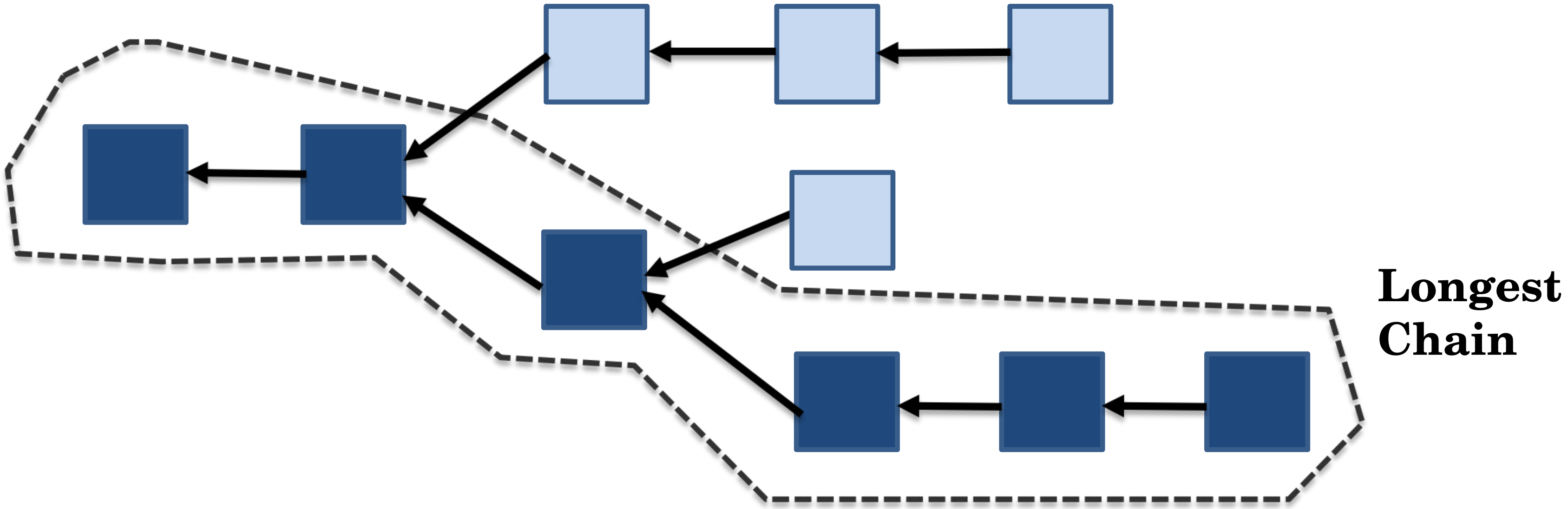
- All chains/forks extend from here

Genesis block is part of the protocol definition

- Hard-coded into each implementation



NAKAMOTO CONSENSUS: DEFINITION



Component 1: (Pseudo-)random Block Generation

- No pre-determined entity generates blocks, but virtually anyone can
- Multiple blocks can be created at the same time

Component 2: Longest Chain Rule

- Correct nodes will always extend the longest chain when creating a new block
- When there are multiple longest chains, pick one at random

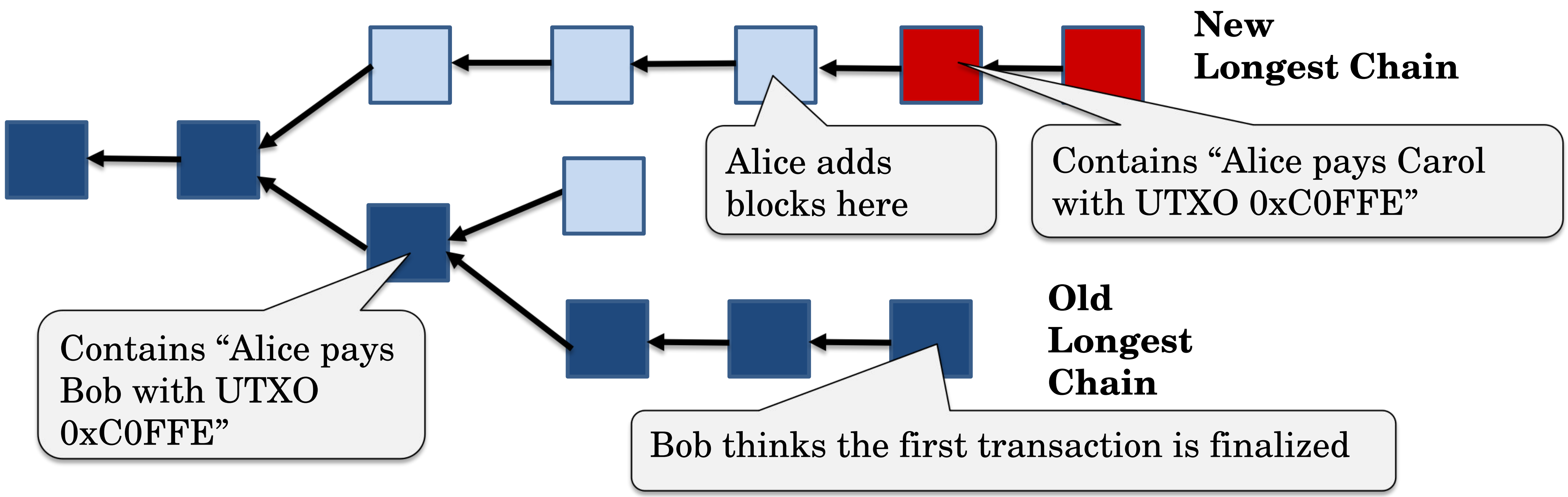
BYZANTINE FAILURES IN NAKAMOTO CONSENSUS

Faulty nodes might

- Not extend the longest chain
- Send invalid blocks
- Create empty blocks
- Delay network messages

What problems could this cause?

DOUBLE SPEND ATTACK



Goal

- When the network assumes a transaction is finalized, create some longer chain that reverts the transaction

NAKAMOTO CONSENSUS: CONVERGENCE

Assumption 1: Synchronous network

- There exists some fixed time bound in which messages, e.g., blocks, will be delivered
- In Bitcoin the bound is usually assumed to be five minutes

Assumption 2: Faulty nodes control a minority of the mining power

- Honest nodes create more blocks on average

At some point it is *virtually impossible* for the faulty chain to overtake the honest chain

```
q=0.3
z=0     P=1.0000000
z=5     P=0.1773523
z=10    P=0.0416605
z=15    P=0.0101008
z=20    P=0.0024804
z=25    P=0.0006132
z=30    P=0.0001522
z=35    P=0.0000379
z=40    P=0.0000095
z=45    P=0.0000024
z=50    P=0.0000006
```

Solving for P less than 0.1%...

```
P < 0.001
q=0.10   z=5
q=0.15   z=8
q=0.20   z=11
q=0.25   z=15
q=0.30   z=24
q=0.35   z=41
q=0.40   z=89
q=0.45   z=340
```

"Figure" from the paper:
q is the mining power of the attacker
z is the length of the “honest” chain

SYBIL RESISTANCE

A malicious entity might try to configure many nodes to take over the network

- These are called *Sybils*
- We need some mechanism to detect or weaken Sybils

In permissioned/private protocols, the members (set of nodes) are pre-defined

- New nodes can only be added with a reconfiguration

In permissionless/public protocols, we need a dedicated Sybil-resistance mechanisms

- e.g., Proof-of-Work (today) or Proof-of-Stake (future lecture)

PROOF-OF-WORK

Goal: Tie likelihood of generating a block to processing power

- Each node only has some finite amount of hardware

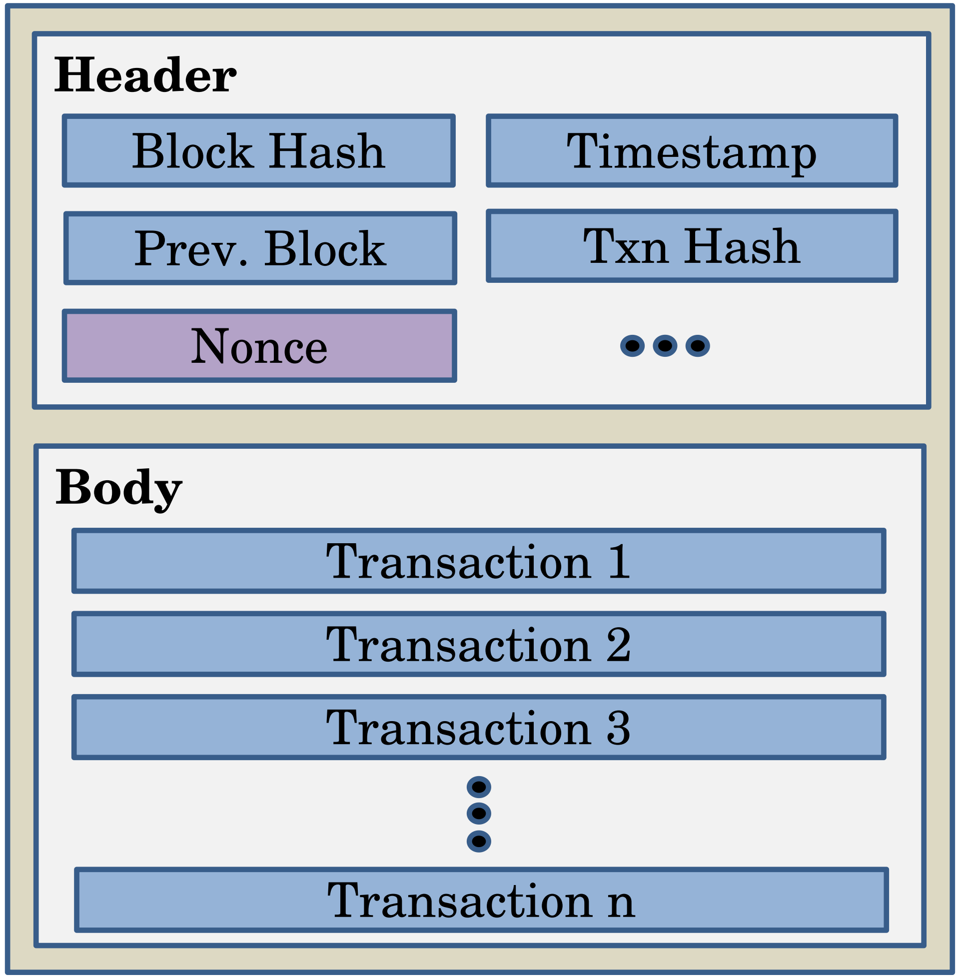
Approach: Create a very hard-to-solve task (the “crypto puzzle”)

- Random tries are needed to find the solution
- We might need many attempts to solve it

hash(Block Data | Nonce) ≤ Difficulty

- Block Data: Set by the miner based on pending transactions, longest chain etc.
- Nonce: Changed by the miner on every attempt
- Difficulty: Set by the protocol

PROOF-OF-WORK IN BITCOIN



- Miners pick a random *nonce* value
- For a block to be valid, the block header's hash needs to be below some difficulty value
- Chance of mining a valid block at any point in time is *independent* of time already spent mining

POW: ENVIRONMENTAL IMPACT

Annualized Total Bitcoin Footprints

Carbon Footprint	Electrical Energy	Electronic Waste
50.83 Mt CO2	91.14 TWh	50.86 kt
Comparable to the carbon footprint of Hungary.	Comparable to the power consumption of Philippines.	Comparable to the small IT equipment waste of the Netherlands.

Single Bitcoin Transaction Footprints

Carbon Footprint	Electrical Energy	Electronic Waste
460.60 kgCO2	825.80 kWh	460.90 grams
Equivalent to the carbon footprint of 1,020,849 VISA transactions or 76,767 hours of watching Youtube.	Equivalent to the power consumption of an average U.S. household over 28.30 days.	Equivalent to the weight of 2.81 iPhones 12 or 0.94 iPads. (Find more info on e-waste here.)

Source: https://digiconomist.net/bitcoin-energy-consumption

BREAK 2

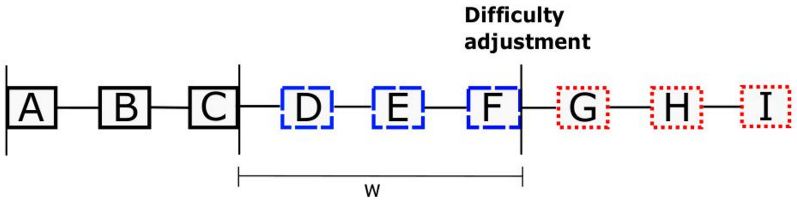
DIFFICULTY ADJUSTMENT

Goal: Ensure that a blocks are created at the same frequency
- e.g., every 10 Minutes in Bitcoin

Challenge: Mining power can change over time
- Miners can join or leave at any time
- Miners might start (or stop) mining if it is (not) economical to so
  - Depends on electricity, cryptocurrency, and hardware prices
- Miners can switch between networks (e.g., from Bitcoin to Dogecoin)

Adjust difficulty based on the *observed frequency vs. the expected frequency*

DIFFICULTY ADJUSTMENT MECHANISMS

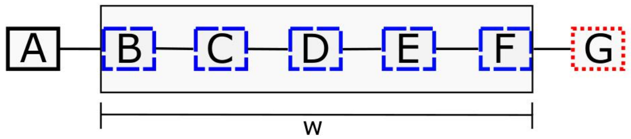


Period-Based

- Every w blocks the difficulty will be adjusted
- For example, in Bitcoin every 2016 blocks (roughly 2 weeks) the difficulty is recalculated

Incrementally Extrapolated

- Every block the difficulty will be adjusted slightly depending on how long it took to mine it
- Difficulty is only adjusted, not recalculated
- Used by Ethereum



Sliding Window

- Every block the difficulty will be adjusted depending on how long it took to mine the last w blocks
- Used by Monero and Bitcoin Cash
- Different implementations have varying window sizes and mechanism to deal with outliers

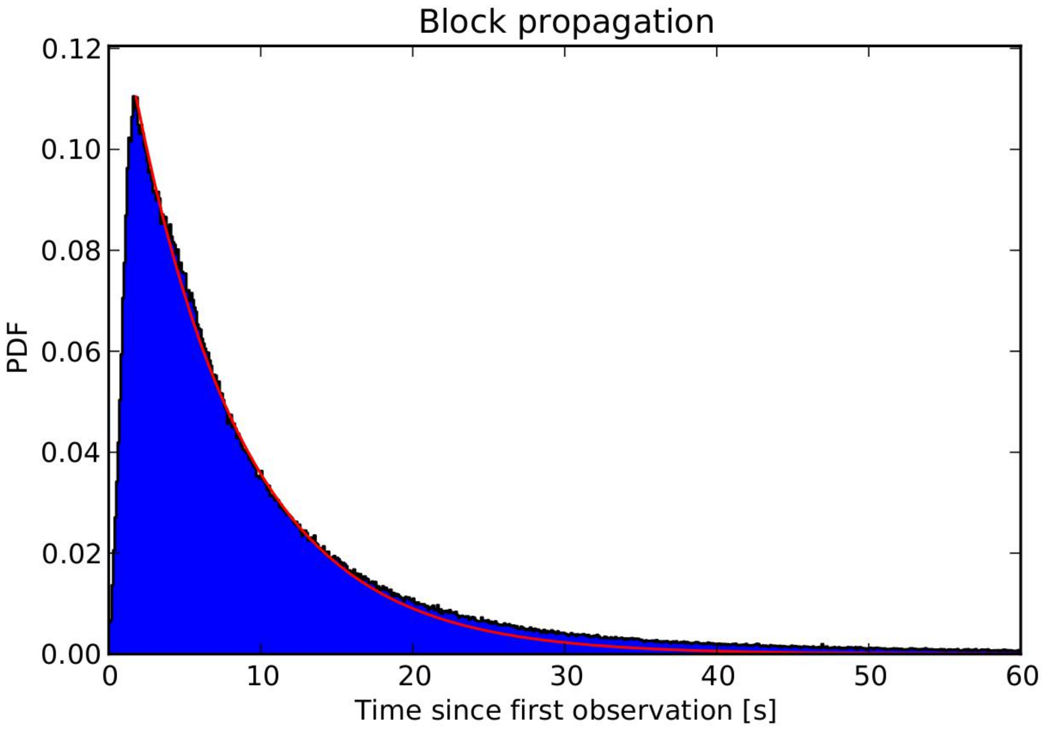
ETHEREUM BLOCK GENERATION

"Information propagation in the Bitcoin network" (Decker and Wattenhofer, 2013)

- 95% of nodes can be reached in <13seconds
- 50% of nodes are reached within 6 seconds
- Numbers might be slightly different today

Why does block propagation take so long?

- Nodes verify/execute blocks before forwarding
- Gossip network introduces additional network hops



THAT'S ALL FOR TODAY

Next time
- More Nakamoto Consensus
- Selfish Mining

Reminder: Project 2a due in a week