

Blockchain-Based Voting System for Secure Distributed Elections

CS 2620: Introduction to Distributed Computing 🎓



Harvard John A. Paulson
School of Engineering
and Applied Sciences

Blockchain-Based Voting System for Secure Distributed Elections

Waseem Ahmad
Harvard College
waseemahmad@college.harvard.edu

Max Peng
Harvard College
mpeng@college.harvard.edu

05/04/25

Abstract

We present a prototype blockchain-based voting system that combines classical distributed-systems principles with modern Web3 tools to deliver secure, transparent, and decentralized elections. Our design builds on an in-class exploration of consensus algorithms, extending our Paxos/Raft/BFT exercises to a Proof-of-Authority (PoA) scheme, and integrates off-chain storage via IPFS alongside Ethereum-style wallet authentication. The system comprises a Next.js client for election management and vote casting, a Node.js/Express server exposing REST endpoints, a custom PoA consensus module in JavaScript, and IPFS for candidate metadata persistence. We validated our approach with end-to-end tests across multiple nodes. Our evaluation demonstrates that PoA reduces computational overhead while ensuring tamper-resistant vote ordering, IPFS guarantees data availability without prohibitive gas costs, and Web3 wallets enable strong voter authentication. This work illustrates how distributed-systems theory can be translated into a real-world application for trustworthy elections, and provides a foundation for further enhancements in scalability, privacy, and more complicated models.

The GitHub repository for this project is available at: https://github.com/waseemahmad1/cs2620_final

Waseem Ahmad | Max Peng

Introduction

- Elections today still rely on physical polling and centralized back-ends.
- This model struggles with accessibility, scaling to large electorates, and transparent audits.
- Distributed systems and blockchain promise a new approach: tamper-resistant ledgers, no single point of failure, and fully verifiable results.

Motivation

- We want to apply CS 2620 principles like consensus, replication, fault tolerance to a real-world voting scenario.
- Blockchain lectures inspired us to explore this system.
- Our goal is a prototype that shows how a decentralized, secure, and transparent voting system could work in practice.

Problem Formulation

Following our proposal, we aim to design a system that:

1. Ensures security via vote integrity
Each vote, once cast, is immutable and tamper-proof through decentralized ledger storage. We plan to evaluate resilience against adversarial attempts and simulated node failures .
2. Promotes transparency while preserving anonymity
All recorded votes are publicly verifiable on the chain and through our audit mechanism, yet voter identities remain anonymous.
3. Eliminates central dependence via fault tolerance
The system does not rely on a single trusted authority for validation or storage. Instead, vote blocks are pinned to IPFS across multiple peers, and block inclusion is managed by a distributed consensus protocol rather than a central server

Our prototype mirrors real-world election systems by combining:

A web-based voting client with Web3 wallet integration

A permissioned PoA blockchain for vote transactions

IPFS for decentralized storage of metadata

A REST-based audit endpoint for independent result verification

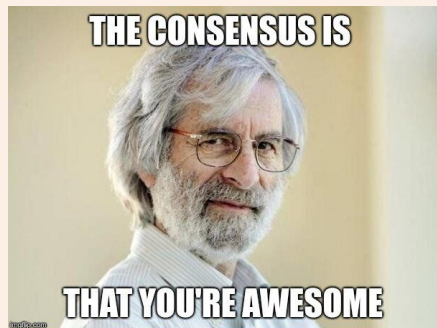
Methodology Overview

System Overview

Our voting platform integrates a Web3 client, a Node.js/Express backend, a private PoA blockchain, IPFS for off-chain storage, and a light Ethereum-style ledger for auditability.

Consensus Algorithm: Proof-of-Authority rotation for fast, permissioned block sealing	Decentralized Storage (IPFS): Pinning of candidate metadata and sealed vote blocks
Ethereum-Style Blockchain: Off-chain smart-contract ledger without on-chain mining	Audit Mechanism: /auditProof endpoint for end-to-end vote verification

Consensus Algorithm



Validators are a fixed set of known identities whose Ethereum-style wallet addresses live in `config/validators.json`

At each round, validators take turns (round-robin) sealing blocks of votes without any mining work

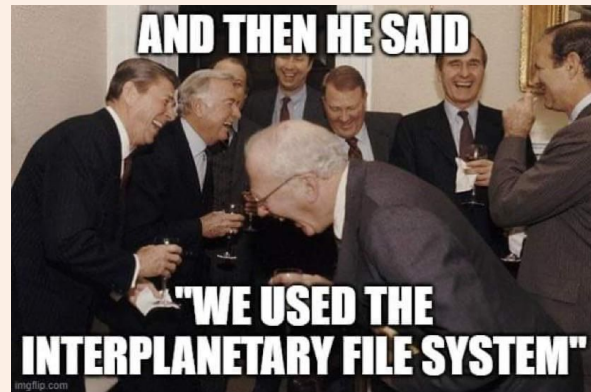
Sealing a block simply means signing the block header (parent CID + new CIDs) off-chain and submitting that signature to the ledger

IPFS (InterPlanetary File System)

Candidate metadata, ballot definitions, and each sealed block's full vote payload are serialized to JSON

These JSON files are pinned to IPFS, producing immutable content identifiers (CIDs)

The blockchain only stores CIDs, keeping on-chain data minimal while ensuring data availability and tamper resistance



Ethereum-Style Ledger

A simple smart contract exposes an `appendBlock` method that validators call off-chain

Each call records the parent CID, new CIDs array, and the validator's anonymized signature

No proof-of-work or on-chain mining; PoA ensures fast finality with minimal gas fees

Full block history is available via JSON-RPC event logs

```
1  const { keccak256, toUtf8Bytes } = require('ethers').utils;
2
3  class Block {
4  constructor(index, timestamp, transactions, previousHash, validator) {
5    this.index = index;
6    this.timestamp = timestamp;
7    this.transactions = transactions;
8    this.previousHash = previousHash;
9    this.validator = validator;
10   this.hash = this.computeHash();
11 }
12
13 computeHash() {
14   const data = `${this.index}${this.timestamp}${JSON.stringify(this.transactions)}${this.previousHash}${this.validator}`;
15   return keccak256(toUtf8Bytes(data));
16 }
17 }
18
19 module.exports = { Block };
```

Audit Mechanism

The `/auditProof` endpoint accepts an election ID and returns all recorded block headers

Server fetches contract events, retrieves each block's JSON from IPFS, and rebuilds vote lists. Ensures transparency and allows everyone to view results

After paying all these gas fees, [#Ethereum](#) logo makes sense to me



Development Log

Most work done in a 9-10 day period.

Days 1-3: Define blockchain data structures
and consensus algorithm

Days 4-6: Integrate IPFS for off-chain storage,
voter authentication with metamask

Days 7 - 10: Work on frontend, write out paper
and presentation

Results

- General System Design
- Design Choices
- DEMO

Authors: Waseem Ahmad | Max Peng

CS 2620: Introduction to Distributed Computing 🐼

Abstract

We present a blockchain-based voting system designed to enable secure, transparent, and decentralized elections. Our implementation features an anonymous voting client, a lightweight consensus protocol, and a verifiable audit mechanism. The system aims to simulate real-world election conditions, showing how distributed technologies can help address issues in modern society (like voting).

Motivation

- Most voting systems today are still centralized and physical (i.e. polling stations and paper ballots)
- These systems face issues of accessibility, scalability, and transparency, leading to raising concerns over election security and voter trust.
- We aim to explore how a **distributed blockchain-based system** can offer **secure, transparent, and decentralized** voting

Tools

- **Next.js + React**: web voting interface
- **JavaScript**: backend logic and vote handling
- **IPFS**: decentralized candidate data storage

Blockchain-Based Voting System for Secure Distributed Elections



Harvard John A. Paulson School of Engineering and Applied Sciences



System Design

- **Hybrid System**: Combines blockchain (Ethereum, IPFS) with centralized authentication
- **Vote Integrity**: Votes stored immutably on ledger system.
- **User-Friendly App**: Built with **Next.js** and **React** for admins and voters.
- **Transparent Results**: Election outcomes verifiable via on-chain data and dashboard.
- **Privacy Protection**: Voter identities stored separately from votes.
- **Decentralized Storage**: Candidate info hosted on **IPFS**.

Mechanisms

Distributed Validation: Vote recording and block creation are handled by multiple validator nodes

Fault Tolerance: The system stays secure and operational even if some nodes fail or act maliciously.

Independent Verification: With a shared blockchain ledger, anyone can verify election results without trusting a central server.



Future Work

- Extend voting to support ranked-choice and multi-candidate elections
- Integrate formal reasoning zero-knowledge proofs to strengthen voter privacy

Acknowledgments

We want to sincerely thank our CS 2620 instructor, Jim Waldo, and the amazing TFs of this class for inspiring and supporting us along the way.

General System Design

Web Client (Next.js)

MetaMask-enabled UI for creating elections, casting votes, ending elections, and viewing the ledger.

API Layer (Node.js + Express)

Multiple server instances run behind a simple HTTP load balancer (`load-balancer.js`). Each instance exposes endpoints like `/castVote`, `/endElection`, and `/auditProof`.

State Synchronization (Redis Pub/Sub)

All servers connect to the same Redis channel. When one instance receives a vote, it publishes the transaction; collaborators' servers subscribe and append the vote to their in-memory chains. Redis also caches recent CIDs for faster lookups.

Blockchain Module (Block.js, Blockchain.js, PoAConsensus.js)

Blocks of votes are sealed via round-robin Proof-of-Authority. Each block header (parent CID + vote CIDs + validator signature) is sent off-chain to our smart contract.

Off-Chain Storage (IPFS)

Candidate metadata, ballot definitions, and each block's vote payload are pinned to IPFS, generating immutable CIDs that the blockchain references.

Ethereum-Style Ledger (Smart Contract)

Validators call `appendBlock` in one lightweight transaction per block—no proof-of-work. Full history is retrievable via JSON-RPC event logs.

← System Walkthrough

Person A

- Start IPFS daemon (`ipfs daemon`)
- Start Redis (`brew services start redis`)
- Launch server on port 3002:
 - `cd server`
 - `PORT=3002 npm start`
- Launch client:
 - `cd client`
 - `npm run dev`
- Start load balancer:
 - `node load-balancer.js`

Person B

- Launch server on port 3003:
 - `cd server`
 - `PORT=3003 npm start`
- (Load balancer already running)
- Launch client:
 - `cd client`
 - `npm run dev`

System Design Choices:

Proof-of-Authority Consensus

We settled on PoA (in `PoAConsensus.js`) instead of PoW because it lets known validators seal blocks instantly without mining.

IPFS for Off-Chain Storage

Putting ballots and candidate metadata directly on-chain would blow up gas fees. Instead, we pin JSON blobs via our `ipfs.js` helpers and only store the resulting CIDs in blocks..

Redis-Backed Multi-Node Sync

To run multiple Express servers behind `load-balancer.js`, we use Redis Pub/Sub (in `index.js`). Whenever one node receives a vote, it publishes to Redis so all instances append the same transaction.

Let's demo our project!!!



Thank you to the CS 2620 Professor Waldo and all the teaching staff for helping make this class an incredible experience!!!

Thank You!

