Secure Voting Using a Permissioned Blockchain

Alec Landow
ajlandow@ncsu.edu
North Carolina State University
Source Code On NCSU GitHub: https://github.ncsu.edu/ajlandow/csc724-adv-distributed-systems

Abstract

The need for a secure method of casting votes that gives voters confidence in election results is vital for the success of a democratic republic. It is important to be able to dispel even the question of such fraudulent activities as ineligible voters voting, ballot duplication, tampering or fabrication, and over-counting of ballots.

I propose using IBM's open source blockchain, Hyperledger Fabric, as the foundation for a decentralized, permissioned voting ledger. The voting ledger will provide the same level of anonymity as public voting registration records, and prevent association of a voter with specific votes cast, as well as prevent the attacks listed above. In addition, any person with the inclination would be able to tabulate the results by communicating directly with the network, without reliance on a third party to do so, although the second party (for example, local government precincts in control of the central nodes that store copies of the blockchain) would be required to give a user authorization to read from the blockchain and to write a vote to the blockchain.

Transactions on Hyperledger Fabric are performed by smart contracts that are implemented as "chaincode" installed on each peer, where both chaincode and the transactions themselves are subject to endorsement based on agreed-upon policies. I have developed a working prototype of a blockchain voting system meeting the above requirements that runs chaincode on multiple distributed peer nodes, and includes auxiliary nodes such as certificate authority and transaction ordering service.

Keywords: blockchain, voting, ballot, bitcoin, ledger, hash, hyperledger, fabric, ibm

ACM Reference Format:

Alec Landow. 2021. Secure Voting Using a Permissioned Blockchain. In *https://github.ncsu.edu/ajlandow/csc724-adv-distributed-systems, CSC724 2021: Advanced Distributed Systems, January 19–May 6, 2021, North Carolina State University.* ACM, New York, NY, USA, 10 pages. https://doi.org/N/A

CSC724 2021, January 19–May 6, 2021, North Carolina State University
2021. ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/N/A

1 Introduction

1.1 Public Blockchains

The popular cryptocurrency Bitcoin[1] uses a public transaction ledger that is a blockchain, a data structure distributed among a large number of interconnected nodes [7]. In the case of Bitcoin, a block is a vector containing information such as transactions of bitcoins, where a bitcoin is a chain of digital signatures [11]. In the context of an electronic voting system, the block would contain cast ballots.

Part of the security of bitcoin comes from the proof of work, which consists of the requirement of completing computationally intensive calculations in order to create a block. The new block creates a hash based on the a hash associated with the previous block. The more blocks created, the more difficult it would be for an attacker to duplicate the chain of hashed blocks. The proof of work makes it very difficult for an attacker to create a phony blockchain as long as more than 50% of the computing power belongs to honest users [9][11].

A general aspect of blockchains, in Bitcoin and elsewhere, is that any transaction or action must be approved by a majority of users. In the case of a public blockchain like Bitcoin, these users are miners, who also run the calculations for the proof of work required to generate a block[10]. In the case of a permissioned blockchain like IBM's Hyperledger Fabric, the endorsement of transactions is more flexible and is subject to agreed-upon policies.

1.2 Permissioned Blockchains and Hyperledger Fabric

Hyperledger Fabric is IBM's open source implementation of a permissioned blockchain[8]. The main documentation describes the blockchain as an immutable ledger, but generally the term "ledger" applies to the blockchain and the "world state" which is equivalent to cumulative results of all transactions on the blockchain. While public blockchains allow anyone to participate anonymously, permissioned blockchains require that the identities of the users be known and subject to permissions that the participants must agree upon in order to participate.

Copies of the ledger are maintained on individual peer nodes, which communicate over what is called a "channel", where a channel consists of a set of permission policies for reading, writing, adding code, and updating permissions. There is one logical copy of a blockchain per channel, and physical copies of the blockchain are stored on each peer node.

Smart contracts, the implementations of which are known as "chaincode" (and the two terms are sometimes used interchangeably), are the distributed packages of code that contain the logic of transacting with the ledger. Chaincode must be approved by all who use it, and any chaincode must be endorsed by the nodes prescribed in the network's policy before that chaincode can be incorporated and used in the network. Each node must install its own copy of a chaincode package.

One or more orderer nodes are responsible for providing consistency of the copies of the blockchains on each of the peer nodes. IBM uses an architecture that it terms as **execute-order-validate**. After a valid chaincode has run and **executes** a transaction proposal (but not yet committed that transaction to its ledger), that transaction proposal is submitted for endorsement to the peer nodes designated in the channel's policy. Each of these peer nodes examines the transaction and sends an approval or denial response to the originating peer.

If the requisite endorsements are present, the originating peer sends the proposed transaction to the ordering service, which receives proposed transactions, stores them in blocks, and collects the blocks into a global **order** using a defined consensus protocol. The default (and only supported in v2.x) consensus protocol for ordering transactions is the Raft protocol[12], which has main characteristics of: strong leadership, randomized-timer leader election, and joint consensus majority overlap during cluster changes. Raft provides fault tolerance that tolerates losing a minority of nodes. Once the order establishes an order of blocks, those blocks are added to the orderer's blockchain.

Subsequently the new blocks and their order are communicated to the peer nodes that are connected to the ordering service, and peers can also transmit these new blocks to other peers via a gossiping protocol. Once a peer receives a block from the orderer (directly or indirectly) the peer **validates** that the transactions in the block contain the requisite endorsements specified by the channel policy, and that new transactions have not been invalidated by recently-added transactions. Once validated, a block is committed to the peer's ledger, and a commit event is published to registered applications.

1.3 Voting System Requirements

I designed an outline for a how the voting system could be implemented in a US presidential election. Each precinct would control one or more peer nodes, one or more orderer nodes, and the precinct would represent the logical "organization" of Hyperledger Fabric. Each state would control one or more admin nodes that would initialize the ballots and policies for all precincts in that state, where all precincts of a state would belong to a statewide Hyperledger channel.

In addition to persisting votes in a secure manner, it is also important that authentication and authorization are performed to ensure each voter is properly registered and eligible to vote. In the United States, voting authentication and authorization laws vary by state. For a voting application to be usable, it would need to allow for this variability and either must be flexible enough to adjust authentication and authorization settings, or it must have scope limited to actions after a person has been authenticated and authorized (by depending on a third party—the Government or legally-valid agent—to do so).

Based on general US election policies, the requirements for the voting system are as follows.

- Voter identification will be publicly known,
- Voters will not be able to be associated with a cast ballot.
- Each voter will only be able to vote once.
- For simplicity, once a vote is cast, it cannot be changed.
- Votes may only be cast within a time window that is specified during ballot initialization.
- The blockchain must be auditable by any citizen of the state.

Figure 1 shows a high level view of what the network might look like in practice.

1.4 Contents of this Paper

The rest of this paper is organized as follows.

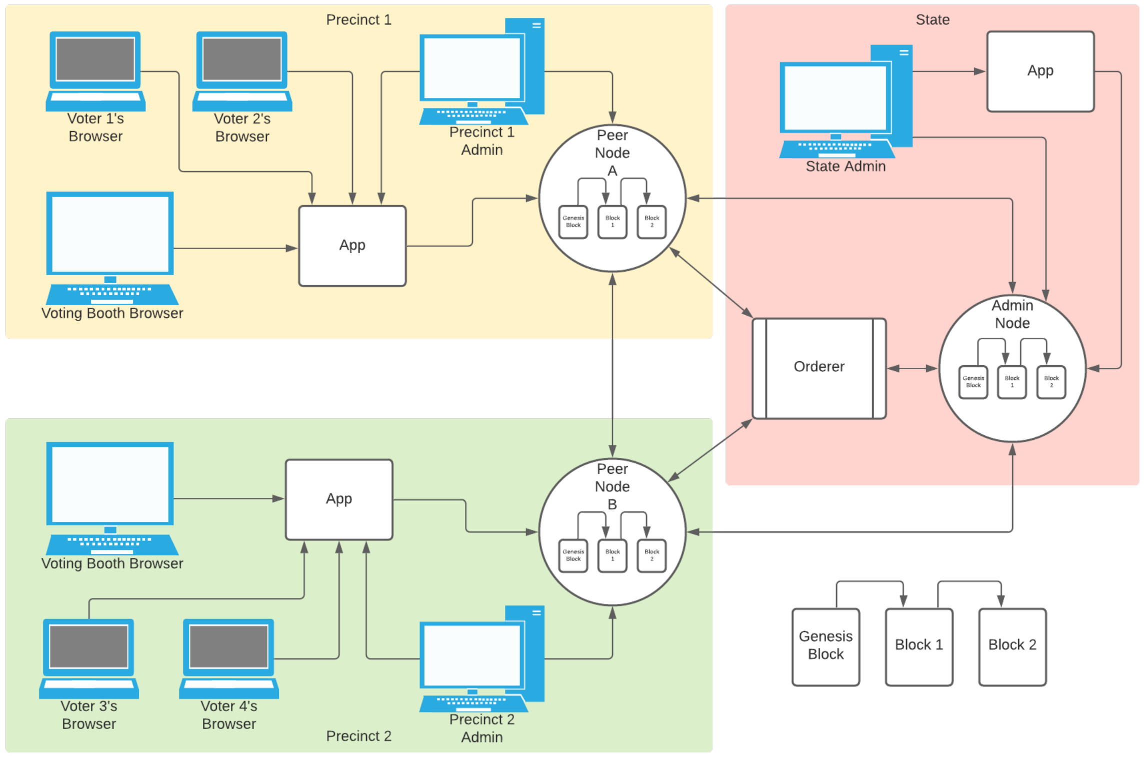

Figure 1. A high level overview of what an implementation of the test network might look like

2 Architecture of the Voting System

The voting system consists of three main layers: the network, the chaincode, and the application. Figure 2 shows how the pieces of the three main layers relate to each other.

2.1 The Network

Since I was not familiar with Hyperledger Fabric when I started this project, I opted to build off of the test network configuration that the Hyperledger developers compiled for education and testing purposes. I used two of the main scripts Hyperledger provided: the first was to start the network, and the second was to add a third peer to the network. However, I had to use the hyperledger command line interface without a script to install and endorse chaincode on the third peer.

Each of the peers belonged to its own organization (org1, org2, or org3), and the single ordering service node also belonged to its own organization (orderer). Each organization had its own certificate authority node, and each of the peer and orderer nodes had its own membership service provider (MSP) implemented on their respective nodes. The membership service providers contained both the private and public keys of the entities with their domains, and were responsible for signing transactions. There was a also a channel MSP that contained permissions information for all nodes in the channel, and copies of this MSP were maintained with a consensus protocol on each of the participating nodes.

The network initialization script allowed for the definition of a channel, which I changed in the script to have the name `test-ballot-channel`.

Each of the nodes was run as a local image within a docker container.

2.2 The Chaincode

Three language SDKs were available for writing chaincode: Java, Go, and Node.js. I opted for Java 8. The chaincode installer required specific build configurations, which I copied from an example chaincode Hyperledger provided. For example, I was unsuccessful in trying to use Maven as a build tool for the chaincode due to configuration issues that were difficult to diagnose. Therefore I used the exact version of Gradle that the example chaincode used, along with a very similar build.gradle file.

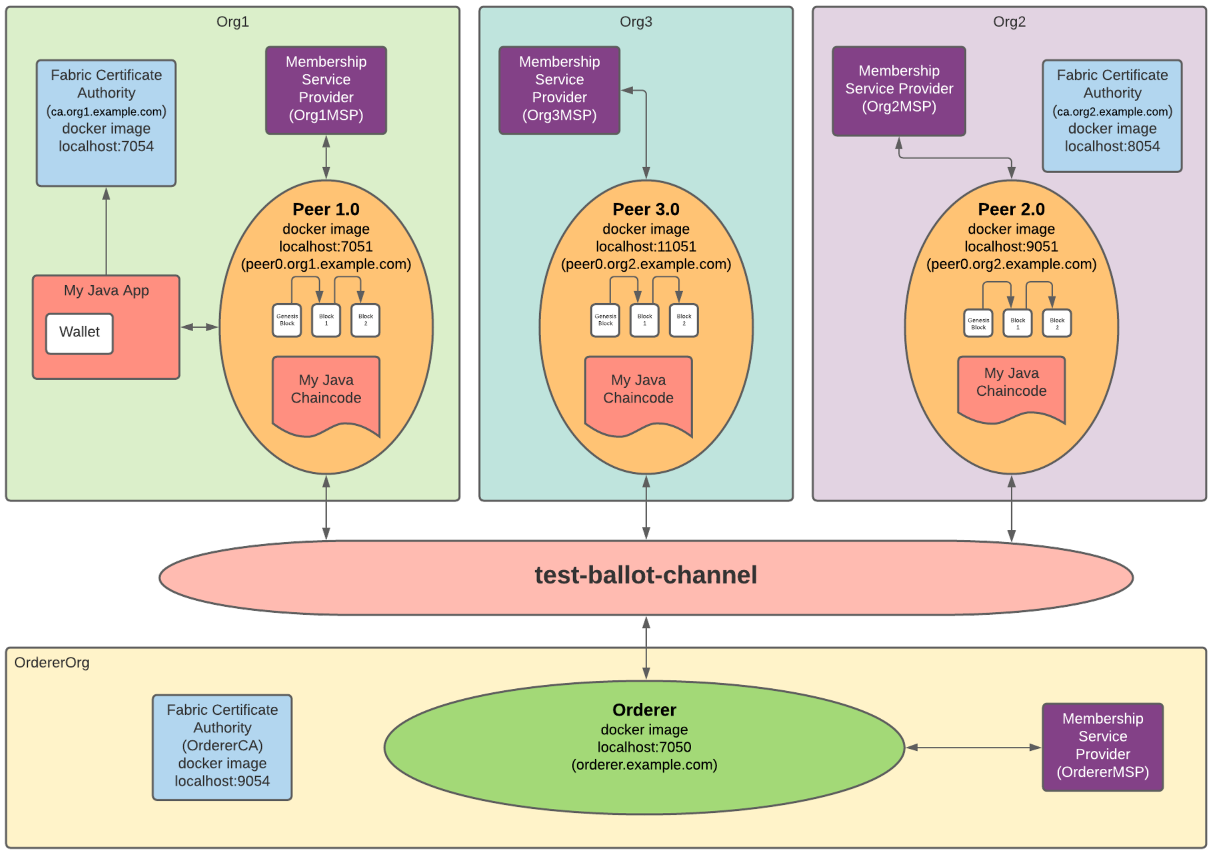
Org1
Fabric Certificate Authority (ca.org1.example.com) docker image localhost:7054
Membership Service Provider (Org1MSP)
Peer 1.0 docker image localhost:7051 (peer0.org1.example.com)
Genesis Block
Block 1
Block 2
My Java App
Wallet
My Java Chaincode
Org3
Membership Service Provider (Org3MSP)
Peer 3.0 docker image localhost:11051 (peer0.org2.example.com)
My Java Chaincode
Org2
Membership Service Provider (Org2MSP)
Fabric Certificate Authority (ca.org2.example.com) docker image localhost:8054
Peer 2.0 docker image localhost:9051 (peer0.org2.example.com)
My Java Chaincode
test-ballot-channel
OrdererOrg
Fabric Certificate Authority (OrdererCA) docker image localhost:9054
Orderer docker image localhost:7050 (orderer.example.com)
Membership Service Provider (OrdererMSP)

Figure 2. The test network

The Chaincode API provided functions for interacting with reading the world state, reading the blockchain, and writing to the blockchain, as well as APIs for examining the certificate of the sender. Data was stored on the blockchain as a set of key-value string pairs, where the value was often stored as a JSON string and deserialized into Java objects. The developers intended for the chaincode to be language agnostic, meaning applications that sent requests to the chaincode could be written in multiple languages, so the arguments passed to chaincode methods had to be passed as strings.

The Chaincode API provided a special annotation, `@Contract`, to define a contract in the code.

Hyperledger defined a lifecycle by which a chaincode could be adopted for use, which can be summarized as follows.

- Package the chaincode (in the case of Java, into a .jar file).
- Install the chaincode on all peers that will use it.
- Submit the chaincode for endorsement to all requisite peers as defined in the channel policy.
- If the endorsements meet those required in the channel policy, then the chaincode is committed to the channel by submitting a transaction to the blockchain.

So not only are data such as votes recorded on the blockchain, but also updates to the code that governs the transactions.

The script to add the third node did not contain the steps to add chaincode, so I had to add the chaincode via the above lifecycle steps using the command line interface Hyperledger provided.

2.3 The App

Hyperledger provided two language APIs for interacting with chaincode from an application: Java and Node.js (and unofficially Python and Go). I again chose Java 8, but could have chosen Node.js since the languages of the application and chaincode are allowed to be different.

The application was responsible for registering users with the certificate authority, maintaining the certificates of users in a wallet, connecting to a single peer using a certificate stored in the wallet, and finally sending requests to a specific chaincode. The requests to chaincode were implemented as methods that took as arguments the name of the chaincode and any string parameters to be passed to the chaincode.

2.4 Casting a Vote (Writing to the Blockchain)

Figure 3 shows the steps taken to write to the blockchain, from a user casting a vote on the app to finally writing that vote to the ledger. The steps follow the general path described in the Introduction. In conducting experiments, I adjusted the endorsement policy to adjust the required number of endorsing peers for approving transactions.

3 Design Goals

3.1 Security

The primary design goal of this project was to build a secure distributed system.

3.1.1 Security Model. The security model is the set of properties assumed to be correct combined with a list of adversaries and their capabilities, or in other words, the trust model combined with the threat model[4].

3.1.2 Trust Model. I assume that a majority of peers are trustworthy and not malicious. The default endorsement policies for approving chaincode and for approving transactions are both "majority", meaning a majority of peers must endorse in order for the artifact to be approved. If the majority of peers were attackers, then the voting system would not function. I also assume that the admin who starts the network is trustworthy, as well as the entirety of the Hyperledger Fabric, insofar as the APIs function as promised.

3.1.3 Threat Model. The peers can also be adversaries. If a political party controls some organizations, and another party controls other organizations, the parties may not always agree on policy changes, chaincode updates, or transactions. Individual voters can be adversaries by trying to vote multiple times, trying to vote if not authorized, or trying to remove or switch votes. Foreign nation states are also a concern.

3.2 Security Methods

Hyperledger Fabric uses standard security practices such as TLS for secure communication and asymmetric keys and certificates to establish identity. The additional security of Hyperledger Fabric comes from the blockchain and smart contracts. The blockchain is an immutable ledger, meaning once a transaction is on the blockchain it cannot be removed. This immutability is enforced not only by the characteristic sequential hashing of blocks, but also by the controlled manner in which the blocks are ordered and added to the blockchain via the ordering service and validation of endorsements.

There are policies at multiple levels, such as the channel, the orderers, or the application, that control access and endorsement procedures. The policies control which organizations can read or write to the ledger, determine which organizations must endorse changes to policies themselves, or to chaincode or updates to the blockchain on a specific channel.

All transactions can be audited in the case of an electoral recount or suspicion of fraud. For example, if a person voted twice be counterfeiting multiple credentials, those credentials would be on the blockchain and would help in determining the legitimacy of a vote. However, since votes are not associated with a voter, the current implementation does not allow for nullifying specific votes.

An important point is that a peer cannot disapprove of a transaction if a vote is for one candidate or another, which is enforced by policy of the voting system that is set at the initialization of the network, for example.

There are some security considerations that are at the border or beyond the scope of the voting system. The manner in which voters are permitted access will have an impact on the integrity of the election. For example, a voter from another state, or a convicted felon, may be allowed (mistakenly or corruptly) to vote. However, the voting system does guarantee that the same credential cannot be used multiple times to vote, since a user's registration information is stored on the blockchain.

3.3 Scalability

For various setups of number of peers and number of endorsers, for each setup I submitted 100 sequential transactions (random votes for one of two possible candidates), using unique test voters each time, and printed the transaction time. The results are in graphs 4 and 5, and will be discussed below.

Scalability was not reliably testable given that the test network had the capability of extending only to three peer nodes and one orderer node. More investigation is needed using a custom and extensible network to evaluate the properties of the voting system as a function of the number of nodes.

3.4 Reliability

Hyperledger has several built-in mechanisms to attain fault tolerance. The Raft protocol of the orderers allows

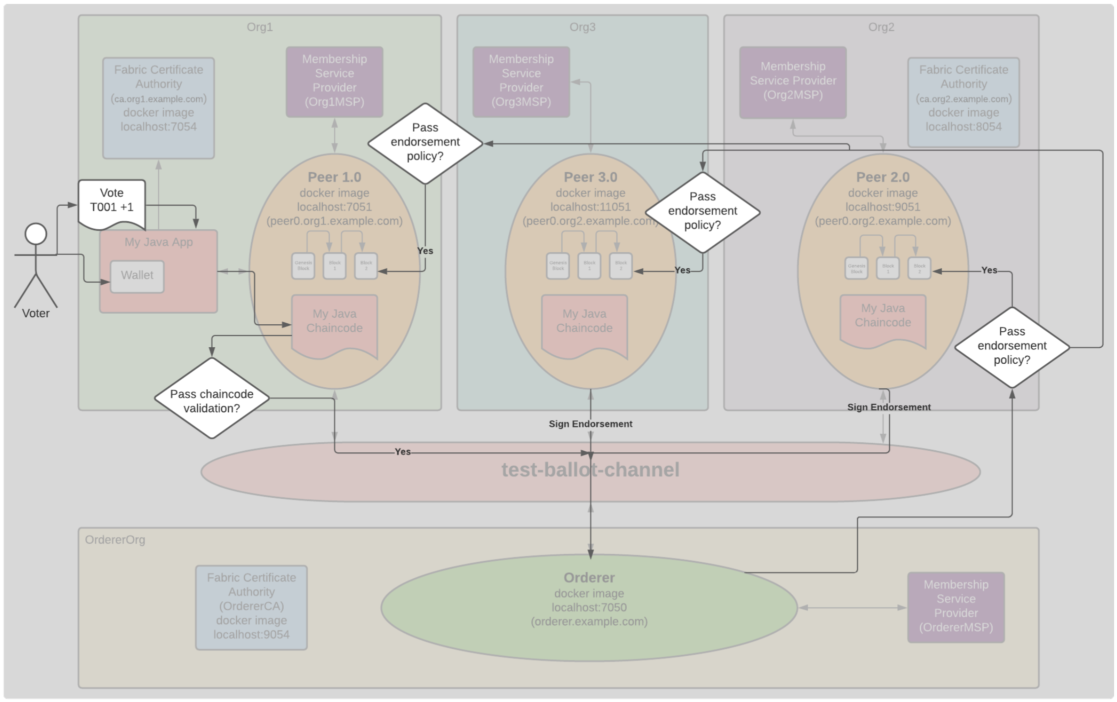
Org1
Org3
Org2
Fabric Certificate Authority (ca.org1.example.com) docker image localhost:7054
Membership Service Provider (Org1MSP)
Membership Service Provider (Org3MSP)
Membership Service Provider (Org2MSP)
Fabric Certificate Authority (ca.org2.example.com) docker image localhost:8054
Pass endorsement policy?
Vote T001 +1
Peer 1.0 docker image localhost:7051 (peer0.org1.example.com)
Peer 3.0 docker image localhost:11051 (peer0.org2.example.com)
Peer 2.0 docker image localhost:9051 (peer0.org2.example.com)
My Java App
Wallet
Voter
Yes
My Java Chaincode
Pass chaincode validation?
Sign Endorsement
test-ballot-channel
OrdererOrg
Fabric Certificate Authority (OrdererCA) docker image localhost:9054
Orderer docker image localhost:7050 (orderer.example.com)
Membership Service Provider (OrdererMSP)

Figure 3. Writing a vote to the blockchain

for a minority of orderer nodes to fail while still maintaining the ability to achieve a consistent global order of blocks. An additional, isolated investigation into the ordering service will be needed to verify the theoretical number of allowable failed nodes.

The distributed nature of the blockchain allows for peer nodes to operate if certain peer nodes fail. Starting with three nodes and a majority endorsement policy (two nodes must approve of a transaction), I tested the voting system by turning off one node (not the node the app was communicating with). I was able to register a user successfully on the blockchain, as well as to cast a vote and read the vote totals. When I removed a second node (also not the node the app was communicating with) I was still able to read the vote totals from the ledger, but I was not able to register or cast a vote, both operations for which two peer endorsements are required. The exception returned by the chaincode to the app was:

```
org.hyperledger.fabric.sdk.exception.
    ServiceDiscoveryException: Failed
     to find any endorsers for
    chaincode basic-ballot
```

3.5 Concurrency

Similar to testing scalability, I submitted 100 transactions (random votes for one of two possible candidates) in parallel, using unique test voters each time, and printed the transaction time. The results are in graphs 4, and will be discussed below. Each test yields only about 10 successful votes cast, and many exceptions like the following were thrown:

```
WARN 111d Block [336] Transaction
    index [8] TxId [53
    e0284faf0b096557adfd3ab
502a4f9e7a67d0c72a4cb7
c0283e5c46ef1912d] marked as invalid
    by state validator. Reason code [
    MVCC_READ_CONFLICT]
```

Based on the only documentation I could find [6], this error appears to be the result of trying to simultaneously update a single key from the application's peer's blockchain. This is consistent with the fact that there are only two possible candidates, so randomly selected one of two candidates would lead to a very high probability of simultaneous update attempts.

Therefore, evidence suggests that parallel updates are not possible on a single node. If multiple voting apps were using a single node, an implementation such as a queue would be needed to ensure only sequential updates.

The above concurrency problem appears when updates are attempted from one peer. When multiple updates are attempted each from different peers, it is the orderer's responsibility to sort near-simultaneous transactions into a single global order. The presence or absence of a queuing system is not mentioned in [8] or [12]. Further investigation is needed with a custom extensible network to determine the concurrency capabilities of simultaneous transactions sent from different peers.

3.6 Openness

A great advantage of Hyperledger Fabric and permissioned blockchains over public blockchains like Bitcoin is that in Hyperledger it is simpler to update the code that supports the network. Not only code, in fact, but also policies. If someone writes chaincode that is endorsed by the peers who will be using it, then that chaincode can then be used on the network without modifying the source code of Hyperledger Fabric.

4 Results

In the test for scalability described in the Design Goals section, I tested sending 100 sequential vote transactions to the ledger using separate networks of two peers and three peers with a policy of majority endorsers (see Graph 4). The average transaction times in each case were 2108 ± 46 ms 2093 ± 15 ms, respectively, within one standard deviation of each other (where the standard deviation is the standard deviation of the average). In each test case the number of endorsers was both two, so the results are consistent with my expectation that the transaction times would be the same.

In a second test, I repeated the first test, but instead of sending the requests in sequence I sent them in parallel all at the same time (see Figure 4). As discussed previously some of these updates succeeded (around 10%), but most failed, leading to undefined behavior.

The third test was to vary the number of endorsers and measure the effect on transaction time. For 1, 2, and 3 endorsers, the average transaction times (still using 100 random vote requests) were 2089 ± 27 ms, 2093 ± 15 ms, and 2125 ± 27 ms, respectively. While there is a small trend of increasing time as a function of endorsers, the times are all within one standard deviation of each other, so I cannot draw any conclusions from this data.

5 Discussion

The tests of scalability led to one expected result and one unexpected result: the transaction time was the same regardless of the number of nodes if the number of endorsers was two, and the transaction time was the same for varying number of endorsers. The former I would expect, since requests would only need to be received from two endorsers in each case in order for the transaction to be validated. However, for the latter my expectation was that the transaction times would increase as a function of the number of endorsers. Potential reasons for this lack of significant time difference could be that the number of endorsers was too small to see a noticeable difference, or that since all nodes were running as local docker images, there were no significant I/O times to magnify the differences.

The multiple levels of security, including the blockchain's hashing, smart contracts, and domain policies, make Hyperledger Fabric a viable candidate to implement a voting system with a small percentage of corrupted votes. However, with so many distributed systems potentially participating, special care would need to be given to a production-level system. The problem of a secure voting system is not only one of technical capability but on policies, especially those policies that operate at the edge of the responsibility boundary of the voting system, such as who is authorized to vote in the system and by whom.

In winner-take-all elections like those in the United States, in which that candidate with the majority, or even the plurality in some cases, wins, there is a strong inclination for two-party systems to develop. If a slight majority of peer nodes favored one party over all others, then scenarios could arise in which policies or transactions in favor of that party could be approved, even with disapproval from the rest of the nodes.

There are two major avenues of future work. The first is to build a custom network using Hyperledger Fabric, and test scalability of peer and order nodes, as well as test concurrent requests from different peers. The second is to develop a set of policies that can maintain the integrity of the blockchain even in the presence of a majority of nodes favoring one party.

6 Related Work

Garay et al. [9] investigated applications that can be run on top of the core of the Bitcoin protocol, which they term the "Bitcoin backbone". The Bitcoin backbone has two fundamental properties: the common prefix property and the chain quality property. The common prefix property essentially states that if a number of

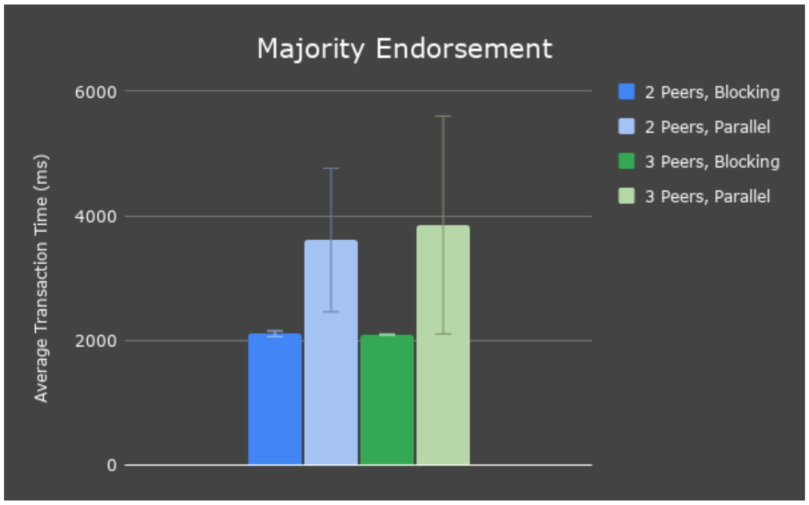

Figure 4

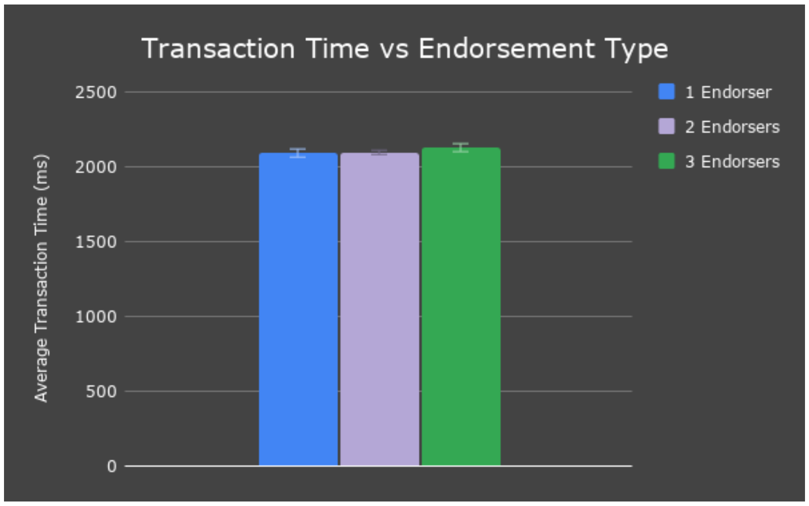

Figure 5

blocks are removed from the end of the blockchain, for a network that synchronizes much faster than the rate of completing a proof of work, the probability that the removed chain will not be a prefix of another honest node's chain is very low. The chain quality property states that as long as the number of honest nodes is greater than 50%, the blockchain of honest nodes are guaranteed to have blocks contributed by honest nodes.

It would be possible to implement a voting system based on the Bitcoin backbone using the interfaces that Garay et al. defined. However, the public blockchain exposes encrypted information unnecessarily to those who do not need it and who might have the resources to break that encryption (such as a nation state)[8]. Additionally the public keys on public blockchains are anonymous, and unless there is a reliable initial validation system to ensure voters are eligible, the public blockchain will not work for a voting system. Additionally, having anonymous public keys means that the blockchain is not auditable, since an auditor will not be able to verify from the blockchain who voted and at what time.

Porutiu et al.[5] have developed a secure electronic voting application that is very similar to the voting system prototype I have built. The app uses Hyperledger Fabric behind Kubernetes, with an interactive web application. Their chaincode appears to perform a similar function as mine in storing on the blockchain whether the voter has voted, as well as storing the election dates.

A small difference I have observed is that multiple elections can exist on the same channel, whereas my implementation would have only one election per channel.

Kang et al. developed a tool called FabZK, which is a Hyperledger extension API for private transactions. My implementation avoided private transactions to maintain auditability. Only certain organizations can view private data on Hyperledger. If there were many private transactions, then only users within participating organizations would be able to see that data. There could potentially be a use case for private data in a voting system, but having channels and controlling channel access already provides a way to isolate data if need be.

Several papers have discussed the precise topic of using a blockchain as the basis for a voting system, and I will discuss some of those here.

Hardwick et al.[7] defined requirements for a secure electronic voting technology, as follows.

- Fairness: no early results are available until all voting ends
- Eligibility: only eligible voters are allowed to vote
- Privacy: the identity of all voters, as well as which vote a voter has cast, will be invisible
- Verifiability: all voters will be able to confirm that their votes have been counted
- Coercion-resistance: this appears to be almost identical to the privacy requirement—specifically, a coercer will not be able to know how a coerced voter has voted
- Forgiveness: the ability to alter a vote after it has been cast

Hardwick et al. state that there already exist commercial remote e-voting companies, among which are BitCongress, Follow my Vote[2], and TIVI[3] (I was unable to find a reliable source for BitCongress), none of which meet all the requirements listed above.

Hardwick et al. created a protocol that meet all the requirements except coercion resistance since they thought it was not possible. In their protocol, the blockchain stores cast ballots, enabling a distributed public database of votes maintained by all users. However, the authors determined that a central authority would be needed to authorize that each user is an eligible voter. The authors proposed a signed eligibility token be used to grant authorization, but they left the exact method of authentication and authorization to the central authority as a functionality that was beyond the scope of what they were trying to do.

The protocol of Hardwick et al. is separated into distinct phases: initialization, preparation, voting, and counting. The very first block contains parameters for the election, and the rest of the blocks contain votes that have unique vote IDs. Each individual voter is responsible for adding a vote to the blockchain, and the rest of the voters have the responsibility of checking to ensure that the voter was authorized and that the voter did not vote multiple times.

Sadia et al. [10], on the other hand, proposed an electronic voting system that is completely decentralized, and relies on Smart Contract to provide security. The authors desribed Smart Contract as an application that runs on top of the blockchain that acts to ensure transactions are secure and free from tampering, although the details are not given. The authors even describe additional authorization of voters using biometrics such as fingerprints.

7 Conclusion

Hyperledger Fabric provides a suitable foundation for a secure voting system due to its security and policy features. The identity of users must be known in a permissioned blockchain, as must be the case for elections in the United States. Smart contracts provide a secure method of reading and writing to the distributed blockchain, and the blockchain itself provides an immutable ledger that can be used to audit elections.

The test network I used contained only three peer nodes, so drawing general conclusions about scalability was difficult. However, the transaction times were consistent with expected behavior as a function of the number of endorsers. I was able to confirm the security policies worked as expected, and that simultaneous updates on a single peer are not possible.

Future work would include implementation of a queue for each node if multiple applications were to vote via that node. Instead of using the test network of Hyperledger Fabric, I would construct a production-level network node by node that would allow for scaling of the number of peer and orderer nodes. Careful consideration must be given to develop network policies that prevent a majority of nodes from corrupting the integrity of the blockchain.

References

[1] 2021. Bitcoin.org. (2021). https://bitcoin.org/en/
[2] 2021. Follow My Vote. (2021). https://followmyvote.com/
[3] 2021. TIVI. (2021). https://www.digitalmarketplace.service.gov.uk/g-cloud/services/638274917583064
[4] William Enck. 2021. CSC574 Computer and Network Security. *North Carolina State University* (Spring 2021).
[5] Horia Porutiu et al. 2019. Build a secure e-voting app. *IBM* (2019). https://developer.ibm.com/technologies/blockchain/patterns/how-to-create-a-secure-e-voting-application-on-hyperledger-fabric/

[6] forum. 2018. MVCC_ READ_CONFLICT when submitting multiple transactions concurrently. *Stack Overflow* (2018). https://stackoverflow.com/questions/45347439/mvcc-read-conflict-when-submitting-multiple-transactions-concurrently

[7] Raja Naeem Akram Freya Sheer Hardwick, Apostolos Gioulis and Konstantinos Markantonakis. 2018. E-Voting with Blockchain: An E-Voting Protocol with Decentralisation and Voter Privacy. (2018). https://arxiv.org/pdf/1805.10258.pdf

[8] IBM. latest. Hyperledger Fabric Documentation. (latest). https://hyperledger-fabric.readthedocs.io/en/latest/whatis.html

[9] Aggelos Kiayas Juan A. Garay and Nikos Leonardos. 2015. The Bitcoin Backbone Protocol: Analysis and Applications. *Proc. Eurocrypt 2015* (2015), 36–44. https://eprint.iacr.org/2014/765.pdf

[10] Rajib Kumar Paul Kazi Sadia, Md. Masuduzzaman and Anik Islam. 2019. Blockchain Based Secured E-voting by Using the Assistance of Smart Contract. (2019). https://arxiv.org/pdf/1910.13635.pdf

[11] Satoshi Nakamoto. 2008. Bitcoin: A Peer-to-Peer Electronic Cash System. (2008). https://bitcoin.org/bitcoin.pdf

[12] Diego Ongaro and John Ousterhout. 2014. In Search of an Understandable Consensus Algorithm. *Stanford University* (May 2014). https://raft.github.io/raft.pdf https://raft.github.io/.