

Faculty of Engineering and Applied Science

Distributed Systems Project Report

**Design and Development of a Blockchain Voting System**

**Team Members**

Omar Shaik, 100591522

Sohaib Mohiuddin, 100593657

Umar Riaz, 100600032

Laxsan Jeyasingam, 100580712

**Professor**: Dr. Q. Mahmoud

[0. Summary 2](#_Toc58296065)

[1. Introduction 3](#_Toc58296066)

[2. Background and Related Work 4](#_Toc58296067)

[2.1 Communication 4](#_Toc58296068)

[2.2 Coordination 5](#_Toc58296069)

[2.3 Consistency and Replication 5](#_Toc58296070)

[2.4 Fault Tolerance 6](#_Toc58296071)

[2.5 Security 7](#_Toc58296072)

[2.6 Bitcoin 8](#_Toc58296073)

[3. Proposed Solution 9](#_Toc58296074)

[3.1 Assumptions 9](#_Toc58296075)

[3.2 Architecture 9](#_Toc58296076)

[3.3 Implementation 9](#_Toc58296077)

[3.4 Scalability 10](#_Toc58296078)

[3.5 Challenges and Solutions 10](#_Toc58296079)

[4. Evaluation and Results 11](#_Toc58296080)

[5. Conclusion and Future Work 14](#_Toc58296081)

[References 15](#_Toc58296082)

# 0. Summary

The purpose of this project is to implement a blockchain based voting system. This project has addressed the major concerns voters have with the traditional election process. The idea was to first validate a user by entering their driver’s license number and allowing them to vote for a candidate of their choice. The vote will then get sent through the client to the NodeServer through the ClientServer. The NodeServer is responsible for creating the block and adding to the block chain. The sender receives the block and multicasts them to the receiver which sends it to the NodeServer. This system aims at making the election process more autonomous and therefore reducing the risk of vote count accuracy, voter authentication. Essentially minimizing the need for humans to be part of the processing of a vote to minimize the risk of human errors. We have accomplished autonomy of processing a vote and saving it to be counted at later stages of the process. Other features we have addressed include reliability of the process through validation as well as consistency as all nodes will receive the updates as the voting process progresses and votes are being added to the blockchain.

# 

# 1. Introduction

Voting allows for groups of people such as communities, tribes, cities, and countries to reach a decision that will represent the choices of a large majority of the members. Voting of a representative occurs after a certain period of time has passed or there is a need to change leadership. When that happens, an election is called, and voters cast their votes to elect a candidate that they believe will represent them the best.

During the election process, multiple processes occur simultaneously to ensure the correct candidate is chosen. The traditional method of voting during elections requires a voter to fulfill multiple criteria to be eligible to vote. For example, in Canada, a voter is required to be a Canadian citizen that is greater than or equal to the age of 18 and must have valid identification. If they meet the above criteria, they are eligible to vote.

Another major process of the election occurs after a voter has casted their vote. The tellers (people who count the votes) need to ensure they are accurate in counting the ballots such as not accidentally skipping one or double counting. This might lead to a candidate getting elected that does not represent the choice of most of the population.

The current election process consists of flaws and major risks that will impact the lives and future of a country and its people. Therefore, in order to minimize these risks, we propose a blockchain based voting system that will address the problems that occur in the current election process such as correct voter validation, accurate count of votes and security during this process.

Our project ensures that only an eligible voter with authentic validation will be able to cast a vote. A voter will be only able to vote once during the election process. Our system also makes sure that all votes are counted accurately, and results are not tampered with. This will ensure that the system set in place is a secure platform that voters can trust during the elections.

The motivation behind the idea of this project is to restore the faith of a country in its leaders and to ensure the leader that is being elected represents and speaks for the interests of much of the population. The blockchain voting system will be a safe, secure, and trustworthy voting platform that will eliminate the major risks that are attached with the traditional election process.

# 2. Background and Related Work

## 

## 2.1 Communication

Communication is probably one of the most important aspects of a connected set of systems or computers. Any distributed system must have powerful and flexible facilities for communication [1]. Distributed systems are expected to offer a higher level of abstraction when it comes to processes communicating with each other [1]. This project looked at a few methods for communication between Client and Server and Master Server to Slave Server(s): RPC, MOM, and Multicast.

Remote Procedure Call, or better known as RPC, is a service that is implemented by a procedure call, as stated in the name. This procedure is executed at the server side. The essence of RPC is transparency, considering its goal is to make synchronous procedure calls look like they are executed locally, when in fact, it is executed on the server side but it is sent back to the client if needed or processed server side for further computation.

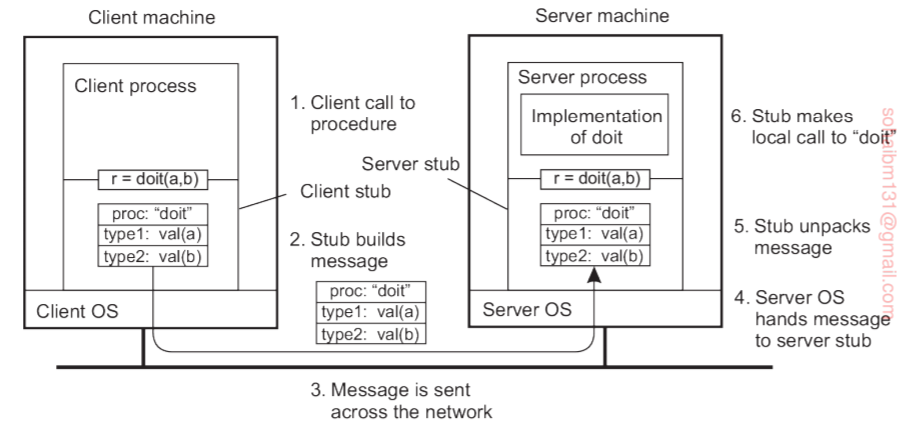


Figure 2.3.1: The steps involved in calling a remote procedure doit(a,b) [1].

The second technique is MOM, or Message-Oriented Middleware. As stated in the name, this is a middleware that allows communication via a message-oriented technique. This technique approaches persistent communication with ease but that means that a queue is involved. Usually, when queues are involved, so is asynchronous communication. The message that is sent by the client is stored by the communication system as long as it takes to deliver it to the server. Since this communication is asynchronous, clients can keep sending messages without stopping [1]. This means neither the client nor the server needs to be executing to receive or send messages [1]. MOM allows for some behavioral similarities to RPC communication such as the ability to synchronize message sending and receiving or blocking sender or receiver while message transmission is in play.

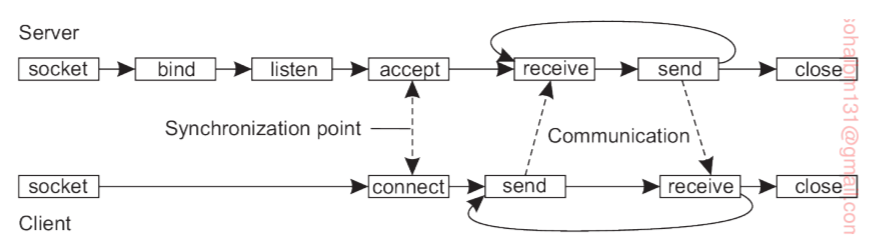


Figure 2.3.2: Connection-Oriented communication pattern using sockets [1].

The third technique is Multicasting. This can be mistaken with Broadcasting but that will be talked about in implementation (Section 3.3). Multicasting allows for one sender to send data to multiple receivers [1]. To visualize what multicasting does, you can picture a tree being created from the sender to all the receivers. The essence of multicasting is to send messages to multiple receivers but also to maintain a robust, dynamic, and synchronous data flow. This encompasses synchronization of data within a set of hosts (both sender and receiver).

## 2.2 Coordination

A distributed system should be able to communicate transparently, but what about communication synchronization? Synchronization is a crucial part of a system depending on what the functionality is. It refers to acting at the right time, which means a host functioning synchronously with the other host(s). The best way to synchronize different systems, especially remote systems, is using clocks to get timestamps to synchronize the data transmission. There are many techniques implemented for synchronization but for the purpose of this project, we considered implementing a Timestamp Server.

Timestamp Servers are used to protect data for long periods of time. Timestamp servers are also widely used in blockchains to provide proof that the data that exists in the linked list is in fact the actual linked list and not a malicious attempt to change the blockchain. Timestamp servers are very specific as they retain an exact moment in time that a certain block has been added or removed or changed (not even a bit can be different).

## 2.3 Consistency and Replication

Replication leads to consistency in most situations, especially when data is in the form of a list and that list needs to be updated on multiple hosts. When a system is distributed, a common area of inconsistencies is the actual data being passed to and from the sender and receiver. This is due to the values being added to the list being either malicious or in the wrong order.

Replication is usually implemented to be distinctive, as such, each replica of the original is associated with the data being propagated, where the data will be propagated, and who propagated the data. With these pieces of data, propagate notifications and operations can be made [1].

## 2.4 Fault Tolerance

A distributed system consists of several independent nodes which process different tasks that interact with each other by means of an interconnecting communication link network which consists of communication components such as modems, Wi-Fi, Ethernet, etc. Unfortunately, the components of almost all systems are naturally imperfect and flawed and therefore are prone to failures which may render the system unable to provide the service or perform the task it was intended to perform. To be able to tolerate the failure of some components, and still be able to keep the service available and running despite these failures. These systems must be equipped with redundancies both in space and time to remain purposeful. Redundant components that take over the part played by failed components. Fault-tolerant distributed computing generally refers to the algorithm which is control of the distributed system’s components to provide the desired service despite the presence of certain failures in the system.

Block chain in most cases uses the Byzantine Fault Tolerance. The Byzantine Fault Tolerance is the characteristic which defines a system that tolerates the class of failures that belong to what is known as the Byzantine Generals’ Problem. Byzantine Failure is the most difficult class of failure modes. Byzantine Faults are the most severe and difficult to deal with. Byzantine Fault Tolerance has been needed in Airline safety protocols, nuclear power plants and all sorts of systems whose actions depend on the results of many sensors. Even SpaceX was considering it as a potential requirement for their systems. An algorithm is said to be Byzantine Fault Tolerant as long as the number of traitors does not exceed one third of the generals. Other variations of the Byzantine Tolerance exist which make solving the problem easier, which include the use of digital signatures or by only allowing communication between certain peers in a network.

How does this all relate to our Blockchain voting system?

Blockchains are decentralized ledgers which, are not controlled by a central authority or location. Due to the value of the data stored in these ledgers, bad actors have huge economic incentives or in our case of a Voting system, political incentive to try and cause faults. With that being said, Byzantine Fault Tolerance, and a solution to the Byzantine Generals’ Problem for blockchains is required.

If there was no Byzantine Fault Tolerance, a node is able to transmit and post false transactions or votes, which would effectively voiding the blockchain’s reliability and purpose. To make things worse, there would be no central authority to take over and repair the damage reliably.

When Bitcoin was invented, a concept of Proof-of-Work was used as a probabilistic solution to the Byzantine Generals Problem as described in depth by Satoshi Nakamoto.

## 2.5 Security

Security in Distributed systems focuses on three main areas:

•Security: to protect the services and data it offers against security threats.

•Confidentiality: the property of a computer system whereby its information is disclosed only to authorized parties

•Integrity: the characteristic that ensures changes to a systems asset can be made only in an authorized way

Some methods for security are

•Encryption: which transform data into something an attacker cannot understand and provides a means to implement confidentiality, it also provides support for integrity

•Authentication: which can verify the claimed identity of a user, client, and server and so on.

•Authorization: check whether the client is authorized to perform the action requested

Cryptography in Block chain

The security of information transmitted from one node to another is questionable, therefore there is a need of using a proper method of transforming it into unreadable formats (secrets writing) through cryptography. The use of a single key or public key cryptographic algorithm which is suitable for protecting message content by hiding information carried by a packet of data during the transmission process. This can be accomplished using cryptography and hashing. Authentication protocols provide a series of communication procedures between users of the system and the server for the purpose of securing the communication process.

Blockchain provides users with data integrity in a place where you cannot really trust anyone, and this is accomplished using cryptography in a way that moves the burden of trust from data processors and third parties to cryptography and hashing algorithms.

Hashing

Blockchains are also dependent on hashing. Hashing is a cryptographic way of converting any kind of data into a string of characters. Along with providing security through encryption, hashing creates a more efficient store of data, as the hash is of a fixed size.

Characteristics of hashing cryptography algorithms

Input must always generate the same output. must consistently produce the same hash with identical characters in the string.

The input cannot be calculated using the output. There should be no way to reverse the hashing process to see the original data set. Not like traditional mathematics.

Change in the input must produce an entirely different output.

The hash should be of a fixed number of characters.

Creating the hash should be fast.

## 2.6 Bitcoin

E-commerce, in practice, relies on central third parties to prevent double spending. There are various issues caused by this, but two main ones are transaction limits and increased costs. So, if a system is needed to bypass these issues, there must be another way to prevent double spending, i.e., if the central party must be replaced.

Bitcoin seeks to provide a potential solution for this through the implementation of blockchain technology and a cryptographic proof-of-work. The proof-of-work is a computationally intensive operation that must be done before blocks can be added, ensuring that as long as the majority of CPU power is controlled by honest nodes, the longest blockchain will be the correct one.

# 3. Proposed Solution

The name of our solution will be called Blockchain Voting System. We decided on this name because the description of the system is within the name itself. We will be implementing a Voting system that uses the help of a blockchain.

## 3.1 Assumptions

* The nodes of the blockchain P2P system are on a secure VPN
* Votes must not be publicly accessible
* The Caster and Listeners provided environment to run continuously

## 3.2 Architecture

The system architecture of this project is a hybrid architecture (partially centralized and partially decentralized). The overall style employed is a combination of layered architecture, object-based and service-oriented architecture, resource-based architecture, and publish subscribe architecture.

The centralized section of the system is the client-server relationship between the permanent nodes of the blockchain and the dynamically added client kiosks set up in various locations for short periods of time and used to vote. The decentralized section of the system is the nodes of the blockchain. Blockchain technology is implemented as a peer-to-peer system and this system is no different. The nodes each have a local copy of the blockchain, and no node has authority over others.

The first style used is the layered architecture. The system is divided into two layers in the shape of a ring. The inner layer consists of the nodes of the blockchain and the outer layer consists of the dynamic clients that attach to those nodes. The second style used is an object-based and service-oriented architecture. This is shown in the logical encapsulation of nodes as separate servers. All servers can have a vote method invoked on them from remote client connections. The clients connect to the nodes through remote procedure calls. The third style used is a resource-based architecture. This is realized in the form of the blockchain stored at all nodes. Of course, not all 4 methods of resource-based architectures are made use of as blockchain technology does not permit deletion, posting, or getting by other processes. It only allows the storage of the blockchain created by listening for broadcasted information. The fourth style implemented is a publish-subscribe architecture. This isn’t realized through the use of message-oriented middleware, but rather through multicast sockets, which are a referentially decoupled and temporally coupled method of sending messages to multiple listeners. The events that the listener is interested in in this case are events casted from a particular address.

## 3.3 Implementation

This project was implemented via ten files: NodeServer.java, ClientServer.java, Implementation.java, Interface.java, Client.java, Sender.java, Receiver.java, Blockchain.java, Block.java, and policy.txt. The core of the system consists of NodeServer objects connected as a peer-to-peer network. A NodeServer has three threads: a ClientServer, a Sender, and a Receiver. Clients connect to a NodeServer’s ClientServer thread and cast votes. ClientServers use Implementation.java to implement the methods they export through Interface.java. Clients also make use of Interface.java to be able to invoke the remote method. Receivers join groups, listen for incoming blocks on those groups, and then sends them to the NodeServer instance to verify then add to its block.

The techniques chosen for the success of this project were RPC and Multicast, specifically, RMI and Server Multicasting. This system is essentially playing the role of both a centralized and decentralized communication system. The communication can be broken down into two parts: Client - Server and MasterServer (sender) - SlaveServer(s) (receivers).

The first part of communication is between client and server. This communication is implemented with Java RMI. RMI allows for objects to be passed between clients and servers in a distributed system with transparency [2]. RMI is very similar to Remote Procedure Call, but the difference lies in the one or more objects being sent along with requests. The transparency is where the objects on both client and server act like local method invocations. Due to the same interface being implemented on both client and server side, this allows for transparency between one or more clients and a server.

The second part is the communication between Master Server and Slave Servers using the implementation of Multicasting. This sounds like broadcasting but is not quite the same. To explain the difference between the two, broadcasting refers to sending data from a host to other hosts in a network, which means that resources must be allocated for computation by the other hosts as well as the main host. Multicasting, however, refers to sending data to other hosts that are part of a group. There may be hosts not part of the defined group of hosts, in that case, those hosts will not have to allocate resources as they are not designated to. A big advantage of using Multicasting to achieve Master Server and Slave Server communication is that there is no restriction on local networks. Multicasting also works on remote networks allowing for synchronized traffic.

## 3.4 Scalability

Our system contains modules that are scalable and can accommodate more voters. The NodeServer will continuously add the newly created blocks that have the voter information in them to the blockchain. We can also increase the number of clients that receive votes from voters through threads. We can also increase the amount of listener nodes and receiver nodes so that we ensure the node is being updated.

## 3.5 Challenges and Solutions

Many challenges were encountered during the implementation of this project. Some of the challenges were related to pass by value and pass by reference. There were many instances during the creation of Block, BlockChain, NodeServer, and Client where variables had to be passed between classes or called remotely, in those cases understanding that Java is pass by value but able to pass by reference gave enough insight into how to statically access the required variables.

Getting to the bigger challenges, number two on the list was the integration of Java RMI and Multicast. The client that would connect to the server to send their vote, would send a value to the server. The server was then to create its own multicast group that would continuously run to receive blocks. We realized that this would cause consistency issues and decided that a hybrid system would work better than fully centralized or decentralized. This meant creating a multicast group of IP addresses that could also communicate with each other as in a P2P communication network.

The hardest challenge of this project was sending the blocks to the blockchain from the receivers. We had already set up a one way linked list that ended at the blockchain being updated on the receivers last. However, that was not the result we wanted. We wanted to be able to send the blocks back and add it to the blockchain as a successful form of consistency and replication. This was meant to be partially transformed in to a peer-to-peer (P2P) system, which was why the block validation was necessary both to and from the receivers (Branches of the tree - tree of sender/receiver nodes).

# 4. Evaluation and Results

Our solution was evaluated through the implementation of the functionality. We envisioned a user story of a voter that will be voting through our blockchain voting system and implemented functionalities as the process advanced. We also performed a series of unit tests along the way to test if each submodule of the project functions as expected according to the user story.

An example of a user story for a vote would be as follows:

A vote is registered to a unique eligible voter. That voter will enter in their driver's licence number to be validated and will choose a candidate. The vote will only be valid for a single vote. The vote will then get sent to the client server and get saved to the system. The vote will travel to NodeServer which will create a new block and add the information in there. Next that block will now be sent to the sender that will send to the receiver. Then the receiver will send to NodeServer to be added to the block chain.

The system was broken down into multiple submodules and each submodule had a dedicated functionality.

Some of these submodules include:

Client: Connects to RMI sends voter information

ClientServer: Handles responses and sends it to NodeServer

NodeServer: holds blockchain. Creates and adds blocks. Calculates proof of work. Sends blocks to Sender

Sender: gets Blocks from NodeServer and multicasts them

Receiver: listens for blocks and sends it to NodeServer

Each of the above submodules were tested independently through unit tests and then we tested the integration of the modules so that each module will work together seamlessly.

Unit tests for Client:Connects to RMI sends voter information

|  |  |
| --- | --- |
| **Test Description** | **Test Result** |
| Successful connection to RMI registry | Pass |
| Receive and save drivers licence number from the voter | Pass |

Unit tests for ClientServer: Handles responses and sends it to NodeServer

|  |  |
| --- | --- |
| **Test Description** | **Test Result** |
| Receives responses from nodes | Pass |
| Sends responses to node server | Pass |

Unit tests for NodeServer: holds blockchain. Creates and adds blocks. Calculates proof of work. Sends blocks to Sender

|  |  |
| --- | --- |
| **Test Description** | **Test Result** |
| Can create a block with given data | Pass |
| Adds block to blockchain | Pass |
| Can hold blockchain | Pass |

Unit tests for Sender: gets Blocks from NodeServer and multicasts them

|  |  |
| --- | --- |
| **Test Description** | **Test Result** |
| Receive block from NodeServer | Pass |
| Sends block to all receiving nodes | Pass |

Unit tests for Receiver: listens for blocks and sends it to NodeServer

|  |  |
| --- | --- |
| **Test Description** | **Test Result** |
| Always listening for newly created blocks | Pass |
| Sends new blocks to NodeServer | Pass |

Integration tests followed once the modules were individually tested for their functionality in order to ensure the information was passing correctly through all the nodes in the system.

The above tests were essential to be carried out since the output of one process is dependent on the input from a separate process. For example, it is essential to save voter information and candidates they voted for to create a block that will have that information and be added to the blockchain. Another major factor is that each process must always be listening for any changes so that the change can be implemented, and all processes would be aware and make their changes accordingly.

# 

# 5. Conclusion and Future Work

By implementing this application, we have successfully accomplished a distributed network that functions as one system. The system automates the voting process from when a voter initially votes to the tallying of votes to finally declaring a leader that has the greatest number of votes. Our implementation is more of a proof of concept that covers the main functionalities of the system and proves that it is possible to implement a blockchain voting system that solves major problems around the election process. There are improvements we could make to this system and add more functionalities to ensure that the system does not require human intervention to help with any of its processes.

A greater focus for improvement will be reliability of the system. Reliability will include security from outside sources, privacy of the voter enables them to vote anonymously and useability of the system.

Our application Blockchain voting has the potential to be implemented for elections that are carried out nationwide. To be prepared for this, the application needs to be scalable. We need to ensure that the system can support a whole nation of voters. Another improvement we can add to the system is in terms of security. The voter will need to authenticate by entering more than just their driver’s license. An example could include biometrics such as scanning of your face to match with the picture on your driver’s license. We will also eliminate the need for accessing the system through the command line and introduce a graphical user interface that will make it easier for any user to navigate through the system.

# References

[1] M. van Steen and A. S. Tanenbaum, Distributed systems. Upper Saddle River: Pearson Educación, 2017.

[2] T. T. Contributors, “What is Remote Method Invocation (RMI)? - Definition from WhatIs.com,” *TheServerSide.com*, 05-Apr-2005. [Online]. Available: https://www.theserverside.com/definition/Remote-Method-Invocation-RMI.

[3] *Bitcoin.org*, 2020. [Online]. Available: https://bitcoin.org/bitcoin.pdf. [Accessed: 08- Dec- 2020].