Pipelined-PBFT Protocol for Banking Transaction Processing

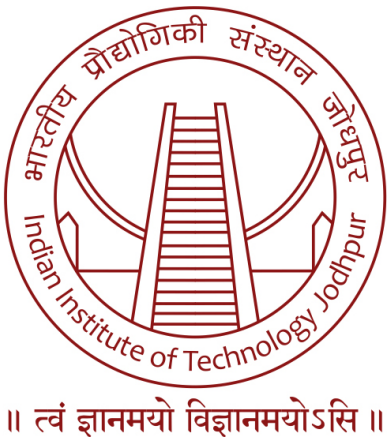
A project report submitted by

Madugunda Jeevan Kumar

MT19CS017

in partial fulfillment of the requirements for the award of the degree of

M.Tech in Computer Science and Engineering



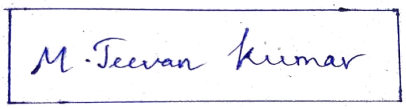
Indian Institute of Technology Jodhpur

Department of Computer Science and Engineering

May 2021

Declaration

I hereby declare that the work presented in this Project Report titled "*Pipelined-PBFT Protocol for Banking Transaction Processing*" submitted to the Indian Institute of Technology Jodhpur in partial fulfillment of the requirements for the award of the degree of Master of Technology, is a bonafide record of the research work carried out under the supervision of Dr. Debasis Das. The contents of this project report in full or in parts, have not been submitted to, and will not be submitted by me to, any other Institute or University in India or abroad for the award of any degree or diploma.



Madugunda Jeevan Kumar
MT19CS017

Certificate

This is to certify that the project report titled "*Pipelined-PBFT for Banking Transaction Processing*", submitted by *Madugunda Jeevan Kumar (MT19CS017)* to the Indian Institute of Technology Jodhpur for the award of the degree of *Master of Technology*, is a bonafide record of the research work done by him under my supervision. To the best of my knowledge, the contents of this report, in full or in parts, have not been submitted to any other Institute or University for the award of any degree or diploma.

Dr. Debasis Das
M.Tech. Project Supervisor

Acknowledgements

I would like to take this opportunity to thank my guide Dr. Debasis Das. He has always guided in the right direction, encouraged me to come up with new ideas.

I would also like to thank Mr. Jayant Vyas, Mr. Ankur Nahar, Mr. Lokendra Vishwakarma for guiding me in choosing the right research path.

I would also like to thank my family and friends too for their moral support and valuable suggestions.

Abstract

The applications of blockchain have been exponentially growing in various domains since its inception in Bitcoin. Initially, people thought that blockchain is used in cryptocurrencies only. Subsequently, many researchers proved that it has potential applications in many industries, financial institutions, academia, health care management, land record management, etc. The consensus protocol is responsible for maintaining the nodes to have the same state of the blockchain. In the past, many consensus protocols like Proof of Work (PoW), Proof of Elapsed Time (PoET), Proof of Stake (PoS), Practical Byzantine Fault Tolerant Protocol (PBFT), etc., were introduced. We are introducing a Pipelined-PBFT protocol which promises to have a lower transaction processing time, higher transaction throughput and less message complexity, when compared with traditional PBFT protocol. Our proposed protocol achieves the speedup of 3, with respect to the conventional PBFT protocol. Our protocol tolerates up to f number of faulty nodes, provided $3f + 1$ number of nodes are present in the network.

Contents

List of Figures

List of Tables

List of Algorithms

1 Introduction & Background

Blockchain is defined as a ledger of transactions that is distributed across the servers/nodes in a decentralized network. A group of transactions are together formed as a block. The blocks are cryptographically linked to each other as shown in Figure 1 meaning the hash of the present block is stored in the next block and hence to tamper any block the attacker needs to make a update through the entire blockchain. Hence, if the record is written into the blockchain, it cannot be tampered. It eliminates the concept of trusting the central authority. This process happens in the decentralised way followed by a consensus process in which all the nodes in a network agree upon the proposed value or the default value. The scenario can be described as there are arbitrarily large number of nodes in the network and all of them has to agree on the value proposed by a honest node even though there are malicious nodes whose fraction is bounded by $f(0 \leq f < 1)$ in the network.

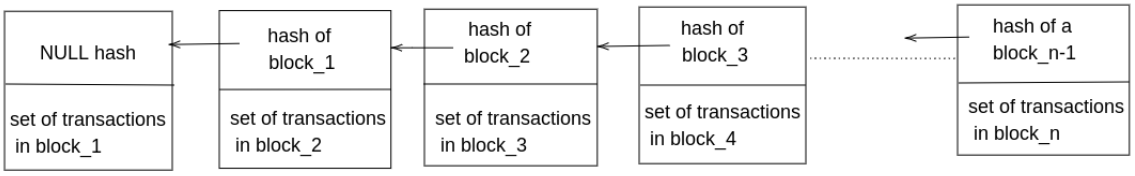


Figure 1: Blockchain

The security properties present in the blockchain are inspiring. The digital payment world has got a lot of impact after the inception of bitcoin. There is a possibility that the applications built on top of the blockchain will revolutionize not only financial service industries but also the non-financial sectors. The blockchain system consists of many technical components. Out of them, distributed consensus protocol is the key component that is responsible for making all the nodes to agree on the same ledger of records/transactions. The choice of consensus protocol can have great impact on blockchain's transaction throughput, scalability, performance and on its ability to be fault tolerant.

The blockchain is classified based on the following categorisation.

1. based on whether the nodes require authorization to participate as a verifier (or validator).
2. based on whether the users can access the data present in the blockchain.

For the first categorisation the blockchain is classified into permissioned blockchain and permissionless blockchain and for the second categorisation blockchain is classified as public blockchain and private blockchain.

Permissionless blockchains:

In this type of blockchain, any node can join the network with the minimum resources required to validate transactions. There is no prior permission required and no central authority to issue any certificate. There might be a concept of incentivisation, where the node is given some kind incentives in the form of cryptocurrency, if the block proposed is found to be valid by all the other nodes in the network. This system is fully decentralised. The major disadvantage of this type of system is that when there is any protocol update, all the nodes have to perform the update and if a few set of nodes doesn't perform the update, then it can lead to forking in blockchain. Eg: bitcoin, ethereum etc.,

Permissioned blockchains:

In this type of blockchain, for any node to join the network, has to go through proper verification by the central authority or the existing group members of the network. This kind of configuration is majorly used in the banking or financial sectors. Based on the specifications of consensus protocol, the number of nodes becoming the verifiers of the transaction may vary. This system may or may not be completely decentralised. The number of nodes in this type of blockchain is restricted and can collectively work together to form a block.

Public blockchains:

In this type of blockchain, anyone can submit or read the transactions that has been written onto the blockchain. For example, a person not in the bitcoin community can see the list of bitcoin transactions that are happening currently.

Private blockchains:

In this type of blockchain, the user cannot read the data present in the blockchain. The verification process and consensus are kept private with respect to a single organisation. Privacy is given much importance in this kind of blockchains. Only the authorised set of nodes within the organisation can read the blockchain. These kind of blockchains are used in banking and finance applications where no one has to know about the transaction details of the customers.

A consensus process is defined as process of achieving necessary agreement on a single state of the network or a single data value in blockchain systems [3]. The consensus mechanism is very important for any blockchain system. Observations on various blockchain consensus protocols showed that a consensus protocol has to be designed by agreeing on a balance among the three objectives: scalability, security and decentralisation [21].

- *Decentralisation*: The system has to be decentralised to the maximum extent possible and must be able to prevent single point of failures.
- *Security:* In the blockchain system, the consensus protocol has to be secure and alive even in the presence of adversary nodes meaning the system must be able to perform the validation correctly and the system must not go down. Finally the blocks which are approved by honest nodes must go into the blockchain.
- *Scalability:* Irrespective of number of nodes in the network, the system must maintain its security level and transaction throughput.
- *Low latency:* The client (or user) waiting time for the transaction approval (or rejection) must be minimal.

Blockchain can be beneficial in the banking sector. One scenario can be, for example, a person 'A' of bank 'B1' wants to transfer some amount to person 'B' of other bank 'B2'. After initiating the transaction, 'B1' validates it and makes the entry in the ledger of 'A', and sends the money to 'B2' by making the entry in the ledger of 'B1'. Upon receiving the money, 'B2' again validates and sends the money to person 'B' by making entry into the ledger of 'B2' and 'B'. This process has got to do a lot of entries in the multiple ledgers involving high friction. For the purpose of this, banks are charging transaction fee for the customers. But in the case of blockchain, ledger entry happens only once (per copy), thus creating less friction. Comparatively, less amount is charged for validation. There can be many more ledger entries that happen for overseas transactions. The idea of using blockchain can be appreciated more in those cases. Many copies of a blockchain can be made, and even if a block of one copy is tampered with, the original block can be fetched from other replicas. In a traditional banking system, all the validations and data storage happens at the central server, and there is a high risk of tampering and crash faults. The applications of blockchain are not confined to banking. It can also be applied in non-banking sectors like submitting and storing the grades in university, asset management, supply chain management, Patient details management in the hospital, etc.

2 Literature Survey

Proof of Work [14] by Nakamoto Satoshi is a consensus protocol used in Bitcoin. Each node in the network groups the transactions into a block and starts solving the hash puzzle in which the hash of the block should contain at least d leading bits. As the hashing is pre-image resistant, the node has to try many times to achieve the target by changing the value of nonce in the block each time as shown in equation 1.

$$Hash(Header \parallel nonce) < target \tag{1}$$

The value of d decides the difficulty of the hash. The more the value of d, the more the difficulty of the hash puzzle. Upon achieving the target, the node declares itself as the winner and broadcasts the block to the network. The other nodes in the network validate the block, and if found valid, they add the block to their respective blockchain. As an alternative to PoW, PoS [15] is proposed by the bitcoin community to decrease the usage of computation power. The weightage is given to the stake held by the node in this consensus process. The node tries to achieve the target for every clocktick as shown in the equation 2.

$$Hash(Header \parallel clock\_time) < target \times stake\_value \tag{2}$$

Here, stake_value refers to the cryptocurrency coins in the blockchain system (here, Bitcoins) invested in participating in the consensus process. For a given node, the probability of proposing a block increases with the increase of the invested coins. Intel proposed PoET [7] as an alternative mining mechanism for PoW in 2016. In this protocol, all the nodes execute a program that makes them stay idle for a random amount of time before broadcasting the block. To ensure that the node truly stayed idle, the node has to perform this process in the trusted execution environment (TEE) called an enclave, where the program execution is resilient to adversary attacks.

The PoW based blockchain system can tolerate up to 50% of the computation power being controlled by the malicious users. To reach the target hash, the node takes a lot of time, resulting in high latency. The time interval between the two block proposals is approximately 10 minutes. To overcome the issue of high computation power and low latency, many alternate algorithms were proposed like bitcoinNG [9], solidus [1], spectre [17], etc. The PoS based blockchain system can tolerate up to 50% of

the stakes being controlled by the malicious users. The malicious users can collude and wait for a long time to acquire the coins and then participate in the PoS protocol to grow a malicious blockchain faster than the honest blockchain. Later many variants were proposed to handle this situation by including the age of the coins. Similar to the PoW, PoET can tolerate 50% of nodes being malicious. The adversary can have more than 50% of the nodes running the program in respective enclaves and can extend the malicious chain. Intel is the attestation service provider and TEE vendor for Hyperledger Sawtooth. As a result, it can be a single point of risk to the blockchain network [21]. It is shown that few attacks in the past have the ability to extract the EPID private key from the hardware. Recent attacks such as cache attacks [5], Foreshadow [18], and Foreshadow-NG [20] have demonstrated the ability to extract the hardware's EPID private key [8]. These attacks become a security threat at the hardware level.

PBFT is the first SMR (State Machine Replication) based BFT consensus protocol proposed by Castro and Liskov [6] in 1999 to make the distributed system byzantine fault-tolerant. The servers connected to each other in a distributed environment can be prone to a byzantine failure. The server goes down accidentally (or deliberately) or approves a block that contains a malicious transaction (s). For a system opting for PBFT protocol to be fault-tolerant, the network should contain minimum of $3f + 1$ nodes, where f is the number of faulty nodes. The PBFT algorithm consists of views divided into three sub-protocols: 1) Normal-operation, 2) checkpoint, and 3) View-change. The Normal-operation is shown in algorithm 1. In the last step of algorithm, the client receives the replies from all the servers and chooses the majority result. The checkpoint protocol keeps track of active transaction requests and helps to discard the older requests. When the Primary node goes down in the view-change protocol, the servers exchange view-change messages with each other, and the next-in-line replica becomes a Primary and continues normal operation. The graphical representation of the PBFT algorithm is shown in Figure 2. The main drawback of PBFT is that it doesn't scale well meaning communication complexity increases as the number of nodes increases and so is the latency time. To improve the scalability, many consensus protocols were proposed by introducing sharding into the networks where the Committees are formed and PBFT protocol is executed parallely in all these Committees.

Algorithm 1 Algorithm for PBFT

Client sends an transaction request

The node with id i is chosen as a Primary node such that $i = v \ mod \ n$, where v is the view number and n is the number of nodes

`/* phase 1: Pre-Prepare phase */`

1 The Primary node generates pre-Prepare message and broadcasts it to the remaining nodes

`/* phase 2: Prepare phase */`

2 Replicas after transaction verification, generates and broadcasts Prepare messages to all the remaining nodes (replicas and the Primary)

3 Each node waits for $\geq 2f + 1$ Prepare messages and enter into Commit phase.

`/* Phase 3: Commit phase */`

4 Servers send Commit messages to each other.

5 Each node waits for $\geq 2f + 1$ Commit messages, and starts to execute the client request and then updates their respective local state

`/* Reply */`

6 Every server replies its result to the client

Elastico [12] is the first sharding-based consensus protocol in the permissionless setting. This protocol aims to parallelize the transaction processing by dividing the network into Committees (or shards) to solve the scalability problem in PBFT. The nodes are grouped into Committees by solving the PoW puzzle. If the Committee size is 'c', the first 'c' nodes that solve the PoW puzzle are formed as directory Committee. The directory Committee then forms the consensus Committees. The consensus Committees then participate in the PBFT algorithm parallelly and send the approved transactions to the final Committee. Upon receiving these transactions, the final Committee gathers these transactions into blocks and sends these blocks to all the Committees. The consensus Committees then receive these blocks and update their copy of the blockchain. The main drawbacks of this protocol are Committee size should be large enough to handle Byzantine faults within the Committee, power consumption is very high due to PoW, there is a possibility of malicious nodes taking control over the directory Committee and stops the entire process of forming consensus Committees, and there is also a possibility of malicious nodes forming the final Committee and stops the process of creating the blocks. SharPer[2] is another sharding-based consensus protocol in a permissioned setting. To meet the drawbacks of Elastico, this protocol assumes that the number of malicious nodes is very less than the total number of nodes present in the blockchain system. [2] assumed that grouping the transactions into blocks can cause latency in transaction processing, and hence they are using one transaction per block. The process of forming Committees is done based on physical location. The nodes at close proximity are formed into Committees, and transactions are divided among these Committees to perform PBFT parallelly. These two protocols successfully achieved good throughput, though the fault tolerance is reduced to

$c/3$ when compared to $n/3$ in traditional PBFT, where c is the Committee size, and n is the total number of nodes. The other most popular consensus protocols that uses sharding are RapidChain [22], OmniLedger [11], Algorand [10], etc. The basic representation of sharding based consensus protocols are shown in figure 3

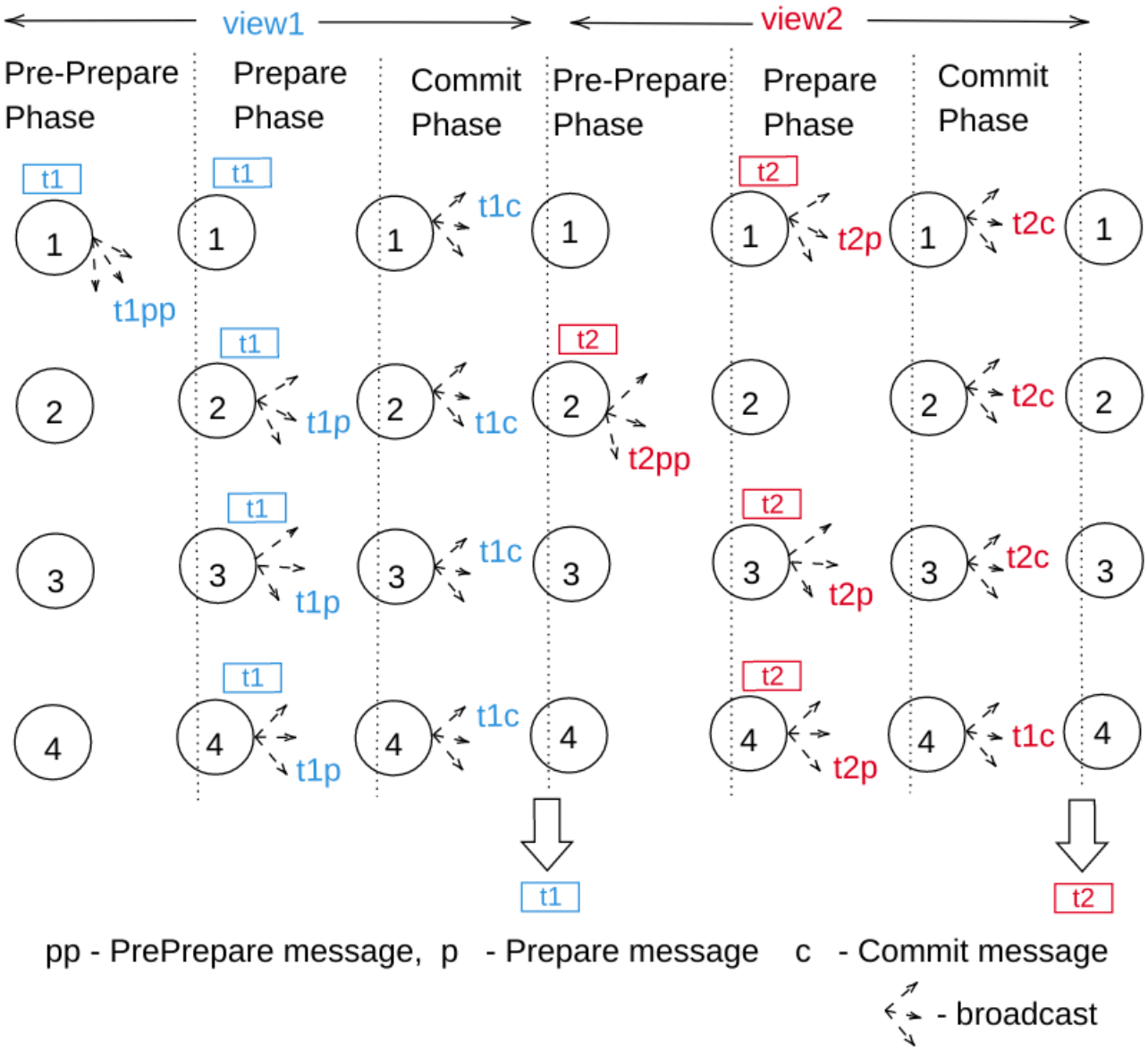


Figure 2: Graphical Representation of PBFT

Prism [4] talks about deconstructing the Nakamoto's blockchain into basic functionalities and scaleup to achieve high throughput. BitcoinNG [9] separates out leader selection from the serialization of blocks by using two types of blocks i.e., key blocks and micro blocks, honey badger of bft [13] for asynchronous networks, Hybrid Consensus [16] involving two or more consensus protocols, Monoxide [19] for scale out blockchains etc.

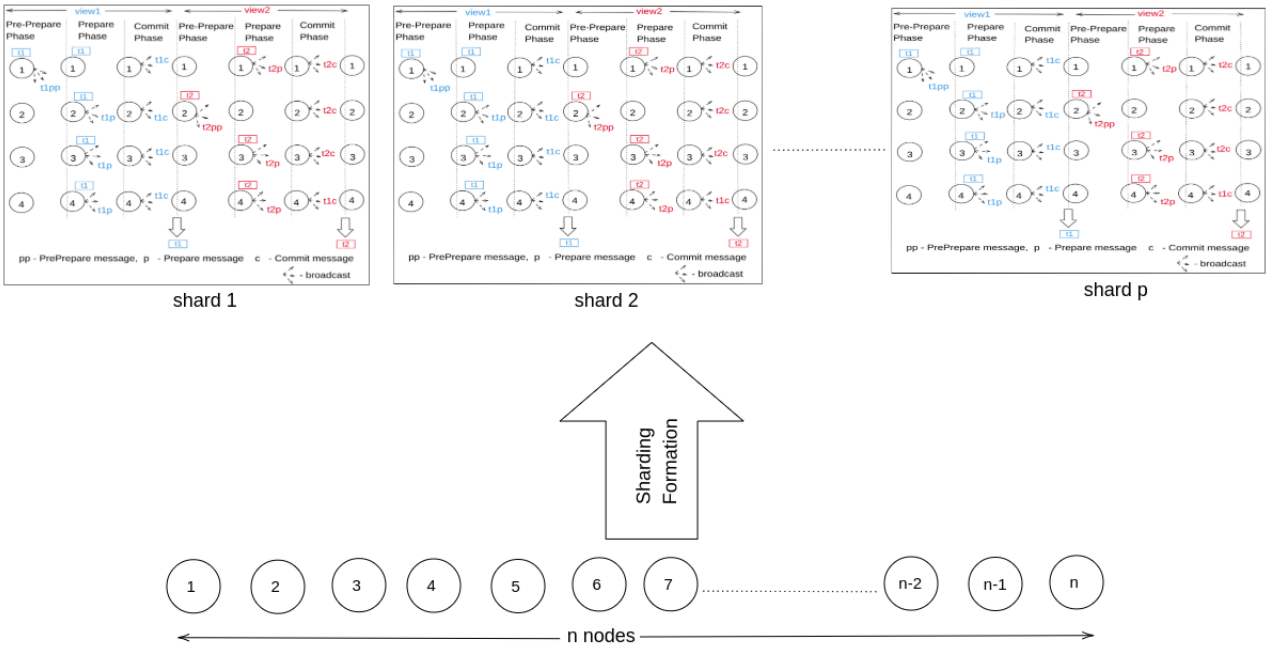


Figure 3: Graphical Representation sharding based consensus protocols

Protocols	Energy Consumption	Forking	Communication Complexity(per block)	Scalability	Fault tolerant
PoW	Very high, when compared to others	yes	$O(n)$	Yes, but with increase in the communication cost	Cannot tolerate more than $n/2$ nodes.
PoS	Depends on the amount of stake held by each node, more the stake, less is the consumption.	yes	$O(n)$	Yes, but with increase in the communication cost	Cannot tolerate more than $n/2$ nodes.
PoET	Very less compared to others, but require trusted environment to execute backoff algorithm.	yes	$O(n)$	Yes, but with increase in the communication cost	Cannot tolerate more than $n/2$ nodes.
PBFT	Less when compared to others	no	$O(n^2)$	Yes, but with increase in the communication cost	Cannot tolerate more than $n/3$ nodes.

Table 1: Comparison Table

3 Problem Definition & Objectives

Our research motivation is directed towards the use of blockchain in the banking and financial sectors. Our work focuses on How to

1. design a consensus protocol that is suitable for banking transaction processing ?
2. improve the scalability issue in traditional PBFT protocol ?
3. avoid a single point of failure, meaning-making the network as decentralized as possible ?
4. design a consensus protocol that is secure even in the presence of adversary nodes ?

Our scheme reduces the number of phases to process the transaction to two by eliminating one round of full broadcast compared to traditional PBFT consensus protocol and by taking advantage of pipelining, the number of phases required to process a transaction is reduced to one. Our protocol is applicable in a permissioned setting and in strongly synchronous network.

Contribution

We introduced Pipelined Practical Byzantine Fault Tolerant protocol suitable for banking transaction processing. In this modified version of PBFT protocol, the functionality of the Pre-Prepare phase is included in the Prepare phase of Pipelined-PBFT eliminating the need for the rudimentary Pre-Prepare phase in PBFT. Moreover, introducing the pipelining concept further maximizes the number of transactions processed within a given time and significantly reducing the message complexity in comparison to the traditional PBFT protocol. In this approach due to pipelining, processing of each transaction takes on average one phase in contrast to traditional PBFT. Furthermore, the modified version guarantees better performance than the traditional PBFT in terms of scalability, transaction processing time, transaction throughput, message complexity and speedup.

4 Proposed Methodology

4.1 System Model

We assume that the entire network system works in permissioned setting, meaning the node needs to go through identity verification to participate in the consensus protocol. All the nodes in the network are connected to each other.

Node failure:
In the real world, systems can experience faults. Majorly the nodes in the network experience two kinds of faults. They are

- *Crash faults:* These faults occur due to power failure, network errors, software errors and due to some other genuine errors.
- *Byzantine faults:* These faults are severer than Crash faults, the nodes might go down deliberately, send faulty messages to sabotage the consensus process.

Network Synchrony:
Network Synchrony deals with the coordination among the nodes in the network. There are three types of network synchrony. They are

- *Synchronous network:* In this network, the messages are sent and received at almost same time. All the participants in the network are coordinated and perform same type of operations with clear time constraints. This can be achieved by having good internet connectivity and centralized clock synchronization service. The network is said to be $\Delta-$Synchronous, if the message delivery is guaranteed within the fixed time delay Δ.
- *Partially Synchronous network:* In this network, the communication between the sender and receiver is guaranteed with uncertain delays and the operations of different nodes are loosely coordinated.
- *Asynchronous network:* In this network, there is no delay guarantee on message communication and the node operations are hardly coordinated.

On that note, we assume that the number of faulty nodes (f) cannot be more than 33% of the total number of nodes (n). Meaning the system should satisfy the condition $n >= 3f+1$. The node commits the transaction only if it receives at least $2f + 1$ Commit messages. We also assume that the network on which we perform the transaction processing is Synchronous network. Table 2 shows the underlying notation of the consensus protocol.

Symbol	**Meaning**
$seqNo$	Sequence number of the transaction
σ_i	Digital Signature by node with id 'i'
p	Primary node
$\langle m \rangle_{\sigma_i}$	Hash value of message 'm' signed by node with id 'i'
$D(m)$	Hash value of the message 'm'
a	Action requested by the client

Table 2: Description table

A typical transaction involving money transfer from one bank account to other can be of the form is shown in table 3.

Sender Account Number	**Receiver Account Number**	**Amount**	**Timestamp**	**Hash** $(s \parallel r \parallel amount \parallel T)$	**Digital Signature (**D**)**
s	r	$amount$	T	D	$\sigma_c(D)$

Table 3: Transaction Request

4.2 Proposed Pipelined-PBFT Algorithm

Pipelined-PBFT is a state machine replication algorithm where the system runs in phases. Replication of servers make the system more robust because the failure of one server doesn't make the entire system go down. The protocol executes in phases. Phase numbers are consecutive in nature starting with 0 and the Primary p is chosen as $\phi \; mod \; n$, where ϕ is the phase number and n is the number of nodes in the network. The algorithm works as follows:

1. Client c submits the transaction request. The transaction request is in the form of $\langle REQUEST, a, T, c \rangle_{\sigma_c}$, where T is a timestamp. Assume that there are other transactions t_{i+1}, t_{i+2}, etc., in the transaction pool in the order of timestamps.

2. The protocol starts as follows.

   (a) The Primary node p of the phase ϕ creates a Prepare message of the form $\langle \langle Prepare, \phi, seqNo, D(t_i) \rangle_{\sigma_p}, t_i \rangle$, where ϕ is the phase number, $seqNo$ is the sequence number. The Primary node broadcasts the Prepare message and enters into the new phase $\phi + 1$.

   (b) The non Primary nodes accept the Prepare message and enter into a new phase $\phi + 1$. The Primary node and the non Primary nodes of this new phase perform various operations.

      i. All the nodes in this new phase $\phi + 1$, validate the transaction t_i and if found valid, creates a Commit message of the form $\langle Commit, \phi, seqNo, D(t_i), r \rangle_{\sigma_r}$, where r is the replica number, and broadcast the Commit messages.

      ii. The Primary node of the current phase along with broadcasting Commit message as in 2(b)i, also broadcasts Prepare message for the transaction t_{i+1} as shown in 2a.

   (c) Upon receiving $\geq 2f + 1$ Commit messages for transaction t_i and Prepare message for transaction t_{i+1}, the nodes enter into phase $\phi+2$. There are different operations for Primary node of this new phase $\phi + 2$ and other non Primary nodes.

      i. The non Primary nodes of this new phase $\phi + 2$ does the following things parallely.

         A. The nodes perform the operation requested in t_i and update the state values accordingly.

B. The nodes create Commit message for transaction t_{i+1} as in 2(b)i, if found valid.

ii. The Primary node of this new phase $\phi + 2$ does the following operations parallely.

A. The node performs the operation requested in t_i and update the state values accordingly.

B. The node creates a Commit message for transaction t_{i+1} as in 2(b)i, if found valid.

C. The node creates a Prepare message for the transaction t_{i+2} as shown in 2a.

(d) Upon receiving $\geq 2f + 1$ Commit messages for transaction t_{i+1} and Prepare message for transaction t_{i+2}, the nodes enter into phase $\phi+3$. There are different operations for Primary node of this new phase $\phi + 3$ and non Primary nodes.

i. The non Primary nodes of this new phase $\phi + 3$ does the following things parallely.

A. The nodes perform the operation requested in t_{i+1} and update the state values accordingly.

B. The nodes create Commit message for transaction t_{i+2} as in 2(b)i, if found valid.

ii. The Primary node of this new phase $\phi + 3$ does the following operations parallely.

A. The node performs the operation requested in t_{i+1} and update the state values accordingly.

B. The node creates a Commit message for transaction t_{i+2} as in 2(b)i, if found valid.

C. The node creates a Prepare message for the transaction t_{i+3} as shown in 2a.

The steps are shown graphically in the Figure 4 and algorithmically in algorithm 2.
The message in every phase is accepted only

1. if the signature of the message is valid.
2. if the digest of the message is correct.
3. if it is not any other message with the same phase number.
4. if the message is received within the threshold time limit.

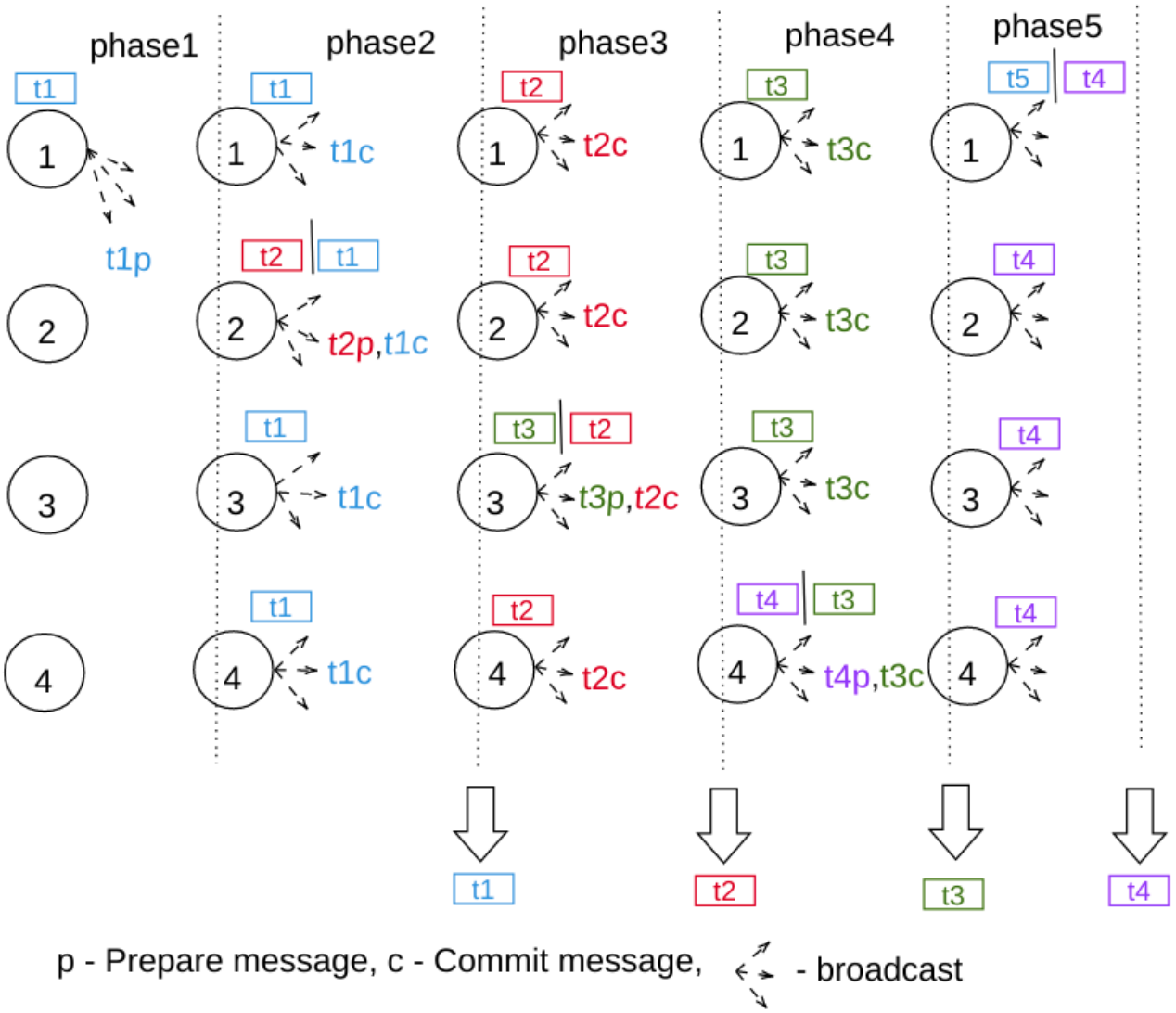


Figure 4: Graphical Representation of Pipelined-PBFT

Algorithm 2 Algorithm for Pipelined-PBFT

The protocol executes in phases, for each phase ϕ the node i is elected as Primary, if $i = \phi \ mod \ n$, where n is the number of nodes in the blockchain network

`/* phase 1: Prepare */`

7 The Primary node broadcasts Prepare message to all the other nodes Replicas record the request

`/* phase 2: Prepare & Commit */`

8 Upon verification of these messages, replicas send Commit messages to all the nodes

9 The Primary node of this phase broadcasts Prepare messages along with the Commit messages

10 Once receiving $\geq 2f + 1$ Commit messages, a server updates local state

`/* Phase 3: Prepare & Commit */`

11 The same operations are repeated as that of in the phase 2

`/* Reply */`

12 Every server replies its result to the client at the end of each phase The client accepts the majority result.

5 Implementation Details

We implemented PBFT consensus protocol in Python (version 3.8.5). We used following modules to implement this consensus protocol.

- **PyCrpytodome**: for generating hash digest for messages and creating public/private keys for the nodes.
- **PyNaCl Library**: for generating digital signatures.
- **Multithreading Package**: for running multiple threads.
- **Pickle module**: for serializing and de-serializing of objects.
- **PIKA module**: a RabbitMQ client for Python.

Subsequently, we implemented Pipelined-PBFT in python using the same version an d same modules that are used for implementing PBFT protocol. Both the protocols are implemented on the same machine. The configurations of this machine are as follows:

- **Processor:** Intel Core i7-9750H CPU @2.6GHz.
- **Operating System:** Ubuntu 20.04.1 LTS, 64bit.
- **Memory:** 15.5 GB.

We implemented PBFT and Pipelined PBFT consensus protocols by implementing state machine using multi-threading paradigm. The pseudo algorithm for PBFT and Pipelined PBFT is shown in the algorithm 3 and algorithm 4 respectively.

Algorithm 3 Pseudo code for PBFT

$n \leftarrow Number\ of\ nodes\ in\ the\ network$
$0 \leq f(Number\ faulty\ nodes) \leq \frac{n-1}{3}$
$node.state \leftarrow STATES[NONE]$
$node.view \leftarrow -1$
while *True* **do**
  if $node.state == STATES[NONE]$ **then**
    $node.view \leftarrow node.view + 1$
    $primary \leftarrow node.view\%n$
    if $node.state == primary$ **then**
      $pre\_preparemsg \leftarrow node.Build\_Pre\_prepare\_Msg(transaction)$
      $node.BroadCastMsg(pre\_preparemsg)$
      $node.state = STATES[PRE\_PREPARE\_SENT]$
    else
      $msg \leftarrow node.consume\_msg()$
      if $Verify\_PrePrepareMsg(msg)$ **then**
        $node.state \leftarrow STATES[PRE\_PREPARE]$
      else
        $node.state \leftarrow STATES[NONE]$
  else if $node.state == STATES[PRE\_PREPARE]$ **then**
    $prepare\_msg = node.Build\_Prepare\_Msg()$
    $node.BroadCastMsg(prepare\_msg)$
    $node.state \leftarrow STATES[PREPARE\_SENT]$
  else if $node.state == STATES[PREPARE\_SENT]$ *or* $[PRE\_PREPARE\_SENT]$ **then**
    $Prepare\_Msgs \leftarrow node.consume\_msg()$
    if $Length(Prepare\_Msgs) \geq 2f + 1$ **then**
      if $Verify\_PrepareMsg(Prepare\_Msgs)$ **then**
        $node.state \leftarrow STATES[PREPARED]$
      else
        $node.state \leftarrow STATES[NONE]$
    else
      $node.state \leftarrow STATES[NONE]$
  else if $node.state == STATES[PREPARED]$ **then**
    $commit\_msg \leftarrow node.Build\_Commit\_Msg()$
    $node.BroadCastMsg(commit\_msg)$
    $node.state \leftarrow STATES[COMMIT\_SENT]$
  else if $node.state == STATES[COMMIT\_SENT]$ **then**
    $Commit\_Msgs \leftarrow node.consume\_msg()$
    if $Length(Commit\_Msgs) \geq 2f + 1$ **then**
      if $Verify\_Commit\_Msgs(Commit\_Msgs)$ **then**
        $node.state \leftarrow STATES[COMMITTED]$
        $CreateBlock(transaction)$
      else
        $node.state \leftarrow STATES[NONE]$
    else
      $node.state \leftarrow STATES[NONE]$

Algorithm 4 Pseudo code for Pipelined PBFT

```
node.state ← STATES[NONE]
node.phase ← −1
t_i ← transaction
i ← 0
if node.state1 == STATES[NONE] and node.state2 == STATES[NONE] then
    node.phase ← node.phase + 1
    primary ← node.phase%n
    if node.id == primary then
        prepare_msg ← node.Build_Prepare_Msg(t_i)
        PrepareLog.append(prepare_msg)
        node.BroadCastMsg(prepare_msg)
        if node.VerifyMsg(PrepareLog[node.phase]) then
            node.state1 ← STATES[NONE]
            node.state2 ← STATES[PREPARED]
    else
        prepare_msg ← node.consume_msg()
        PrepareLog.append(prepare_msg)
        if node.Verify_PrepareMsg(PrepareLog[node.phase]) then
            node.state1 ← STATES[NONE]
            node.state2 ← STATES[PREPARED]
while i ≤ (no_of_transactions − 1) do
    i ← i + 1
    if node.state1 == STATES[NONE] and node.state2 == STATES[PREPARED] then
        node.phase ← node.phase + 1
        primary ← node.phase%n
        if primary == node.id then
            commit_msg = node.Build_Commit_Msg(PrepareLog[node.phase − 1])
            CommitLog.append(commit_msg)
            prepare_msg = node.Build_Prepare_Msg(t_{i+1})
            PrepareLog.append(prepare_msg)
            node.BroadCastMsg(CommitLog[node.phase − 1], PrepareLog[node.phase])
            commit_msgs ← node.consume_msg()
            if Length(commit_msgs) ≥ 2f + 1 then
                if node.Verify_Commit_Msgs(commit_msgs) then
                    node.state2 ← STATES[COMMITTED]
            if node.Verify_PrepareMsg(PrepareLog[node.phase]) then
                node.state1 ← STATES[PREPARED]
        else
            commit_msg ← node.Build_Commit_Msg(PrepareLog[node.phase − 1])
            node.BroadCastMsg(commit_msg)
            commit_msgs ← node_consume_msg()
            if Length(commit_msgs) ≥ 2f + 1 then
                if node.Verify_Commit_Msgs(commit_msgs) then
                    node.state2 ← STATES[COMMITTED]
            if node.Verify_PrepareMsg(PrepareLog[node.phase]) then
                node.state1 ← STATES[PREPARED]
    else if node.state1==STATES[PREPARED] and node.state2==STATES[COMMITTED] then
        node.state1 = STATES[NONE] and node.state2 = STATES[PREPARED]
```

6 Comparison Analysis

We performed comparison between PBFT consensus protocol and Pipelined PBFT protocol in terms of transaction processing time, transaction throughput, message complexity per transaction. We also introduced another parameter called speedup, which measures the relative performance between the two entities (here PBFT and Pipelined PBFT).

6.1 Transaction Processing time and Transaction Throughput

Initially we started testing average transaction processing time for PBFT protocol and Pipelined PBFT protocol over the set of nodes ranging from 5 to 150. We performed these experiments on over the transactions ranging for 10 to 10000. The results are obtained as shown in the Figure 5. We also performed experiments for transaction throughput and observed the results as shown in the Figure 6.

We observed that transaction processing time increases with the increase in number of nodes and transaction throughput decreases with the increase in the number of nodes in case of both PBFT and Pipelined-PBFT resulting in the latter performing better than the former. This is due to, Pipelined-PBFT processes transactions in two phases for first transaction and transaction processing is done in one phase there after with the help of pipelining. Our protocol also performs well in terms of scalability, when compared with PBFT.

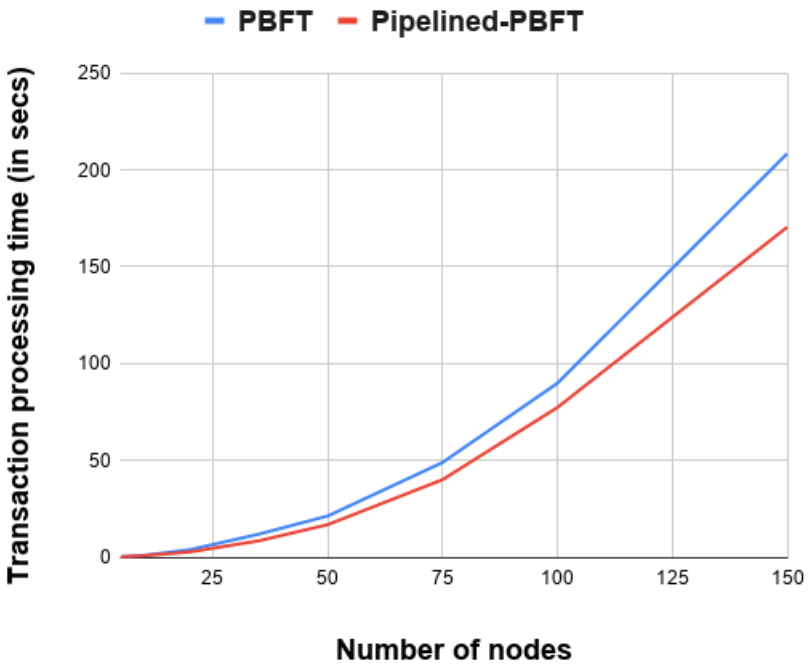


Figure 5: Transaction processing time between PBFT and Pipelined-PBFT

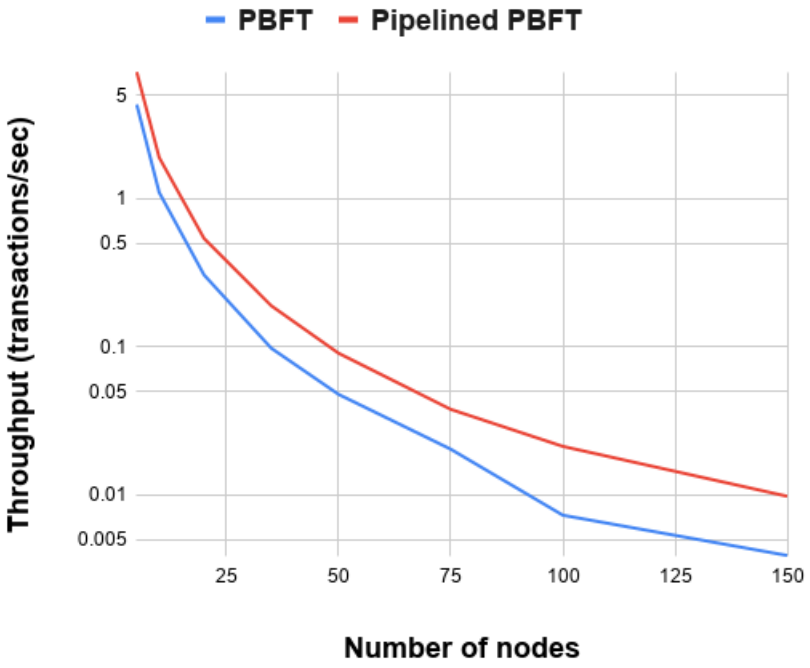


Figure 6: Throughput comparison between PBFT and Pipelined-PBFT

6.2 Message complexity per transaction

The main drawback of PBFT and Pipelined-PBFT is that the message complexity increases with the increase in number of nodes (replicas). But Pipelined-PBFT tends to have less effect when compared to PBFT.

In PBFT, considering there are 'n' number of nodes in the network and 'f' number of faulty nodes, where $f \leq (n-1)/3$, the minimum and maximum message count for processing one transaction in the Pre-prepare phase are $(n-1)$ and $(n-1)$ respectively, the minimum and maximum message count for processing one transaction in Prepare phase are $(n-1)*(n-f-1)$ and $(n-1)*(n-1)$ respectively, the minimum and maximum message count for processing one transaction in Commit phase are $(n-1)*(n-f)$ and $n*(n-1)$ respectively. The details are shown in the table 4.

	Pre-Prepare	**Prepare**	**Commit**
Minimum Message count	$n-1$	$(n-1)*(n-f-1)$	$(n-1)*(n-f)$
Maximum message count	$n-1$	$(n-1)*(n-1)$	$n*(n-1)$

Table 4: The message count for processing one transaction in PBFT

The lower limit on the total number of messages (M) used for processing one transaction in PBFT is $2 * (n-1) * (n-f)$, and that of upper limit is $2 * n * (n-1)$ and is shown in the given equation 3.

$$2 * (n-1) * (n-f) \leq M \leq 2 * n * (n-1),\ where\ f \leq (n-1)/3 \tag{3}$$

In Pipelined PBFT, considering there are 'n' number of nodes and 'f' number of faulty nodes, where $f \leq (n-1)/3$, the minimum and maximum message count for processing one transaction in Prepare phase is $n-1$ and $n-1$ respectively, the minimum and maximum message count for processing one transaction in Commit phase is $(n-1) * (n-f)$ and $(n-1) * n$ respectively.

	Prepare phase	**Commit phase**
Minimum Message count	$n-1$	$(n-1) * (n-f)$
Maximum Message count	$n-1$	$(n-1) * n$

Table 5: The message count for processing one transaction in Pipelined PBFT

The lower limit on the total number of messages (M) for processing one transaction in Pipelined PBFT is $(n-1) * (n-f+1)$, and the upper limit is $(n-1) * (n+1)$ and is shown in the equation 4.

$$(n-1) * (n-f+1) \leq M \leq (n-1) * (n+1),\ where\ f \leq (n-1)/3 \tag{4}$$

The graph depicting the comparison of message count for processing one transaction is shown in the figure 7 and figure 8.

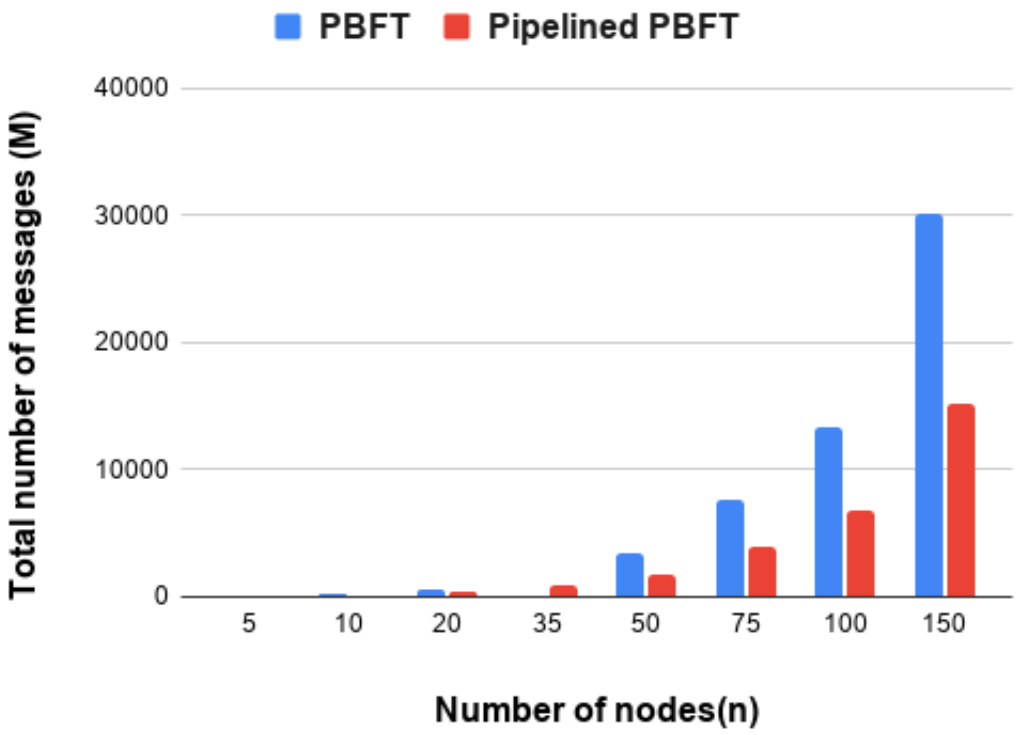


Figure 7: Minimum number of messages for processing one transaction

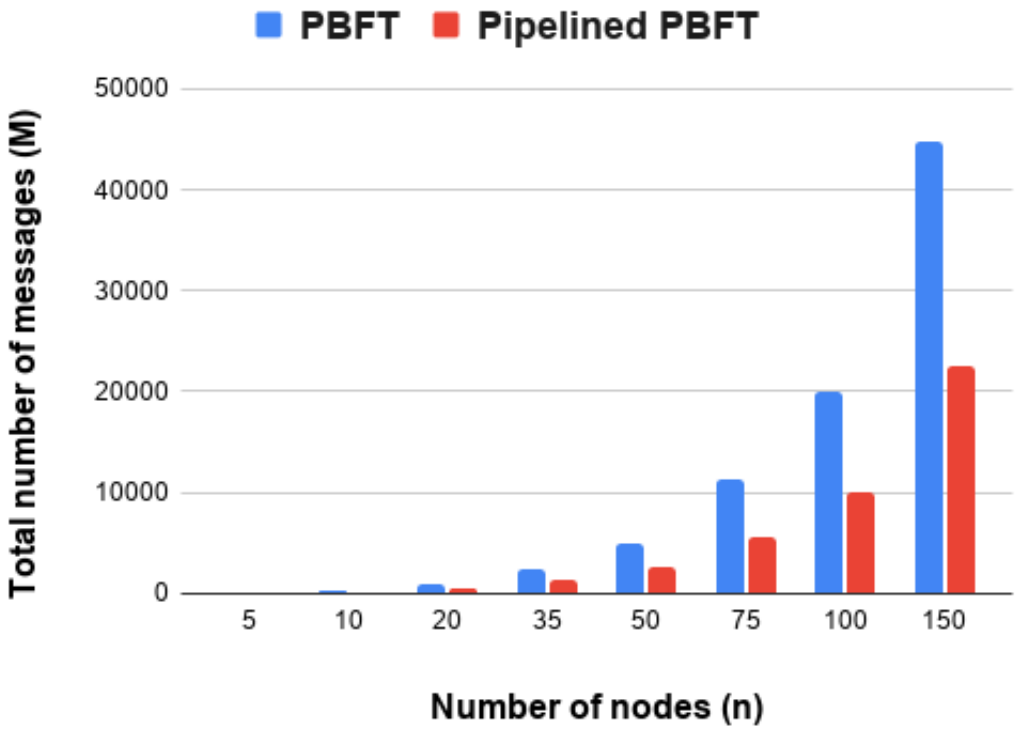


Figure 8: Maximum number of messages for processing one transaction

6.3 Speedup

Speedup is defined as the measure of relative performance between the two processes performing same operation. In PBFT, assume each phase takes time of t units. Each transaction takes $3t$ units. For 'm' transactions, it takes $3mt$ units of time. In Pipelined PBFT, for the first transaction, it takes two phases and one phase for the rest of transactions. Assuming each phase takes $(t + \delta)$ units, where δ is very small. The first transaction takes $2(t + \delta)$ units and rest of the transactions take $t + \delta$ units ideally. Hence 'm' transactions take $2t * 1 + t(m - 1)$ equals $(m + 1) * (t + \delta)$. Hence the speedup of Pipelined PBFT is given in equation 5.

$$Speedup\ of\ Pipelined\ PBFT\ = \frac{Time\ taken\ to\ process\ transactions\ using\ PBFT}{Time\ taken\ to\ process\ transactions\ using\ Pipelined\ PBFT}$$

$$Speedup = \frac{3mt}{(m+1) * (t+\delta)}$$

$$Speedup \approx \lim_{\delta \to 0} \frac{3mt}{(m+1) * (t+\delta)}$$

$$Speedup \approx \frac{3m}{m+1} \tag{5}$$

Assume we have very large number of transactions to process. Then the speedup is shown in the equation 6.

$$Speedup \approx \lim_{m \to +\infty} \frac{3m}{m+1}$$

$$Speedup \approx \lim_{m \to +\infty} \frac{3}{\frac{m+1}{m}} \tag{6}$$

$$Speedup \approx 3$$

Denial of service attack occurs when the adversary takes the control of one of the nodes and doesn't broadcast Prepare message for these transactions. In this scenario, the primary node of the next phase after waiting for time 't' units, broadcast the Prepare message for the unsettled transaction. For every

transaction that is involved in the DOS attack, it takes additional 't' to process the transaction. Assuming Denial of service attack is possible for $x\%$ of total transactions (m). The modified speedup is shown in the equation 7.

$$Speedup\ of\ Pipelined\ PBFT\ = \frac{Time\ taken\ by\ PBFT\ under\ DOS\ attack}{Time\ taken\ to\ by\ Pipelined\ PBFT\ under\ DOS\ attack}$$

$$\begin{aligned} Speedup &\approx \frac{x*m*(\gamma*t+3*t)+(100-x)*m*3*t}{x*m*(\gamma*t+2*t)+(100-x)*(m+1)*t} \\ Speedup &\approx \frac{x*m*\gamma*t+300*m*t}{x*m*\gamma*t+x*m*t+100*m*t-xt} \\ Speedup &\approx \frac{x*\gamma+300}{x*\gamma+x+100} \end{aligned} \tag{7}$$

$\gamma(1 \leq \gamma \leq \frac{n-1}{3})$ is the parameter that indicates how many nodes halts the transaction consecutively.

1. *Case 1 ($x = 0$):* This is the case of maximum achievable speedup.

$$Speedup \approx \frac{0*\gamma+300}{0*\gamma+0+100} = 3$$

2. *Case 2 ($x = 100, \gamma = 1$):* This is the case of minimum achievable speedup due to DOS attack and network failures.

$$Speedup \approx \frac{100*1+300}{100*1+100+100} = 1.33$$

3. *Case 3($x = 50, \gamma = 1$):*

$$Speedup \approx \frac{50*1+300}{50*1+50+100} = 1.75$$

7 Conclusion

In this project, we have introduced a new consensus protocol for banking transaction processing. It is a two phase (Prepare, Commit) protocol. The two phases are pipelined such that the first transaction alone is processed in two phases while the subsequent transactions are processed at every phase.The Primary node in any phase generates Prepare message and broadcasts it to the network. The nodes then move to the new phase. The non Primary nodes in this new phase validate the transaction present in the Prepare message and generate Commit message. Subsequently, the generated commit messages are broadcasted over the network. The primary node of this new phase generates prepare message and broadcasts to the network. In addition to the above action, the primary node validates the transaction present in prepare message and broadcasts the commit message over the network parallely similar to the functioning of non-primary nodes. Upon receiving $\geq 2f + 1$ Commit messages and one Prepare message, the nodes move to the new phase. The above procedure is iterated from there on. We have implemented both PBFT and Pipelined-PBFT algorithms and analyzed the results. The results undoubtedly indicated that the Pipelined PBFT delivered better performance over the conventional PBFT algorithm in terms of transaction processing time, transaction throughput and message complexity per transaction. However, few drawbacks involved with the proposed Pipelined PBFT cannot be denied.

Drawbacks of the proposed Pipelined-PBFT consensus protocol are:

1. For a distributed network system, maintaining strong synchrony is cost oriented. However, the proposed pipelined PBFT approach is highly reliant on the synchrony of the whole distributed system.

2. The Pipelined PBFT consensus protocol is used only in permissioned blockchain setting.

3. The nodes in the network should move from one phase to other phase almost simultaneously. To achieve this, the communication among the nodes in the network has to be fast and strong.

4. At all the times, atleast $2f + 1$ nodes should broadcast the messages (prepare or commit) to make sure that the network tolerates 'f' faulty nodes.

8 Future Work

The focus of our current research is to adapt the concept of blockchain into banking and financial sectors. In the wake of the same, we have come up with a new consensus protocol called Pipelined PBFT which is suitable for banking transaction processing. This improvised algorithm promises to perform better than traditional PBFT in terms of transaction processing time, transaction throughput and message complexity per transaction. However, few drawbacks involved with the proposed approach cannot be denied. Our future research work focuses on

1. overcoming the drawbacks of our proposed Pipelined PBFT consensus protocol.
2. introducing super scalar pipelining in the Pipelined PBFT consensus protocol to achieve still more better performance.
3. introducing blockchain features in other financial applications like cheque processing, loan processing, insurance processing, credit/debit card services etc.,
4. extending our consensus protocol to other non financial sectors like health care, academia, land record management, IOT etc.

References

[1] Ittai Abraham et al. “Solidus: An incentive-compatible cryptocurrency based on permissionless byzantine consensus”. In: *CoRR, abs/1612.02916* (2016).

[2] Mohammad Javad Amiri, Divyakant Agrawal, and Amr El Abbadi. “On sharding permissioned blockchains”. In: *2019 IEEE International Conference on Blockchain (Blockchain)*. IEEE. 2019, pp. 282–285.

[3] Andreas M Antonopoulos. *Mastering Bitcoin: unlocking digital cryptocurrencies*. ” O’Reilly Media, Inc.”, 2014.

[4] Vivek Bagaria et al. “Prism: Deconstructing the blockchain to approach physical limits”. In: *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*. 2019, pp. 585–602.

[5] Ferdinand Brasser et al. “Software grand exposure:{SGX} cache attacks are practical”. In: *11th {USENIX} Workshop on Offensive Technologies ({WOOT} 17)*. 2017.

[6] Miguel Castro, Barbara Liskov, et al. “Practical Byzantine fault tolerance”. In: *OSDI*. Vol. 99. 1999. 1999, pp. 173–186.

[7] Lin Chen et al. “On security analysis of proof-of-elapsed-time (poet)”. In: *International Symposium on Stabilization, Safety, and Security of Distributed Systems*. Springer. 2017, pp. 282–297.

[8] Ittay Eyal. “Blockchain technology: Transforming libertarian cryptocurrency dreams to finance and banking realities”. In: *Computer* 50.9 (2017), pp. 38–49.

[9] Ittay Eyal et al. “Bitcoin-ng: A scalable blockchain protocol”. In: *13th {USENIX} symposium on networked systems design and implementation ({NSDI} 16)*. 2016, pp. 45–59.

[10] Yossi Gilad et al. “Algorand: Scaling byzantine agreements for cryptocurrencies”. In: *Proceedings of the 26th Symposium on Operating Systems Principles*. 2017, pp. 51–68.

[11] Eleftherios Kokoris-Kogias et al. “Omniledger: A secure, scale-out, decentralized ledger via sharding”. In: *2018 IEEE Symposium on Security and Privacy (SP)*. IEEE. 2018, pp. 583–598.

[12] Loi Luu et al. “A secure sharding protocol for open blockchains”. In: *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*. 2016, pp. 17–30.

[13] Andrew Miller et al. “The honey badger of BFT protocols”. In: *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*. 2016, pp. 31–42.

[14] Satoshi Nakamoto. *Bitcoin: A peer-to-peer electronic cash system*. Tech. rep. Manubot, 2019.

[15] Cong T Nguyen et al. “Proof-of-stake consensus mechanisms for future blockchain networks: fundamentals, applications and opportunities”. In: *IEEE Access* 7 (2019), pp. 85727–85745.

[16] Rafael Pass and Elaine Shi. “Hybrid consensus: Efficient consensus in the permissionless model”. In: *31st International Symposium on Distributed Computing (DISC 2017)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik. 2017.

[17] Yonatan Sompolinsky, Yoad Lewenberg, and Aviv Zohar. “SPECTRE: A Fast and Scalable Cryptocurrency Protocol.” In: *IACR Cryptol. ePrint Arch.* 2016 (2016), p. 1159.

[18] Jo Van Bulck et al. “Foreshadow: Extracting the keys to the intel {SGX} kingdom with transient out-of-order execution”. In: *27th {USENIX} Security Symposium ({USENIX} Security 18)*. 2018, pp. 991–1008.

[19] Jiaping Wang and Hao Wang. “Monoxide: Scale out blockchains with asynchronous consensus zones”. In: *16th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 19)*. 2019, pp. 95–112.

[20] Ofir Weisse et al. “Foreshadow-NG: Breaking the virtual memory abstraction with transient out-of-order execution”. In: (2018).

[21] Yang Xiao et al. “A survey of distributed consensus protocols for blockchain networks”. In: *IEEE Communications Surveys & Tutorials* 22.2 (2020), pp. 1432–1465.

[22] Mahdi Zamani, Mahnush Movahedi, and Mariana Raykova. “Rapidchain: Scaling blockchain via full sharding”. In: *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*. 2018, pp. 931–948.