TELKOMNIKA Telecommunication, Computing, Electronics and Control
Vol. 18, No. 4, August 2020, pp. 2224∼2234
ISSN: 1693-6930, accredited First Grade by Kemenristekdikti, No: 21/E/KPT/2018
DOI: 10.12928/TELKOMNIKA.v18i4.15701

Fair and trustworthy: Lock-free enhanced tendermint blockchain algorithm

Basem Assiri, Wazir Zada Khan
Department of CS and IS, Jazan University, Jazan, Saudi Arabia

Article Info

Article history:

Received January 30, 2020
Revised April 13, 2020
Accepted April 29, 2020

Keywords:

Blockchain protocol
Consensus protocols
Correctness
Cryptocurrency
Fairness
Lock-free
Smart contracts

ABSTRACT

Blockchain Technology is exclusively used to make online transactions secure by maintaining a distributed and decentralized ledger of records across multiple computers. Tendermint is a general-purpose blockchain engine that is composed of two parts; Tendermint Core and the blockchain application interface. The application interface makes Tendermint suitable for a wide range of applications. In this paper, we analyze and improve Practical Byzantine Fault Tolerant (PBFT), a consensus-based Tendermint blockchain algorithm. In order to avoid negative issues of locks, we first propose a lock-free algorithm for blockchain in which the proposal and voting phases are concurrent whereas the commit phase is sequential. This consideration in the algorithm allows parallelism. Secondly, a new methodology is used to decide the size of the voter set which is a subset of blockchain nodes, further investigating the block sensitivity and trustworthiness of nodes. Thirdly, to fairly select the voter set nodes, we employ the random walk algorithm. Fourthly, we imply the wait-freedom property by using a timeout due to which all blocks are eventually committed or aborted. In addition, we have discussed voting conflicts and consensuses issues that are used as a correctness property, and provide some supportive techniques.

Corresponding Author:

Wazir Zada Khan,
Department of CS and IS,
Jazan University,
Main University Campus, Jazan, 45142, Saudi Arabia.
Email: wazirzadakhan@jazanu.edu.sa

1. INTRODUCTION

Blockchain technology, also called Distributed Ledger technology is capable of achieving and maintaining data integrity in distributed systems, so the interest in this technology has been increased. The term blockchain was started in the 90's as variant of distributed database in which event or information was stored. After the global financial meltdown in September 2008, the term "Bitcoin" was first introduced by Satoshi Nakamoto [1]. Being the first application as a pure peer to peer electronic payment concept that was initially proposed to avoid the incurred cost caused by online payments. During the period of September 2014-2015 [2], bitcoin has experienced an exponential growth with over 4 million users, which has transformed it into one of the most powerful computing networks in existence. After the success of Bitcoin, blockchain has been applied in many applications including smart cities [3], content delivery networks [4], access control systems [5], smart grid powered systems, Internet of Things (IoT) and Industrial IoT (IIoT) [6, 7].

User transactions can be recorded in a decentralized fashion rather than centralized digital ledger approaches. A blockchain avoids the alteration of all subsequent blocks or collusion of all nodes in the network by maintaining a distributed and decentralized ledger of records across multiple computers[8]. The transactions and some details (such as sender, receiver, amount of money, and hash number) are grouped into datasets referred to as ”blocks” [9]. The blockchain users are called ”miners” who record and share the data blocks over the whole blockchain network. Every node has a copy of the ledger and all node are connected for communications. The nodes communicate to maintain the copies of the ledger and to synchronize processes. Through integration of edge computing and blockchain technology, high throughput and efficiency of transaction processing can be achieved while maintaining data security and integrity [10].

The operation and working of blockchain is as follows: A new transaction record is created when a blockchain user performs a transaction which is then transferred to neighboring peer users in the blockchain network. Prior to adding blocks to the public ledger, verification of transactions is done by the neighboring peer users who used to perform mining by utilizing consensus algorithms. These distributed consensus protocols are used to ensure that the newly added transactions are not altered or not fake. For the validation of the block, the nodes in the peer to peer network solve a crypto-puzzle called Proof-of-work (PoW) by using computational resources at their disposal [11]. Since this approach is considered to be an energy inefficient consensus approach, in order to avoid depletion of computational resources, various consensus mechanisms are proposed including proof-of-stake (PoS), proof of activity, proof of space, variants of Byzantine fault-tolerant algorithms (BFT). The proof-of-stake (PoS) consensus protocol is an interestingly attractive one because it does not cost computing power and provides increased protection from a malicious attack on the network. In PoS mechanism, there is pre-distribution of stakes at the beginning of the process. The entities that have stakes in the system are allowed to take block-inclusion decisions, irrespective of blockchain’s length or history of the public ledger.

Differing in the consensus mechanism, recently many variants of blockchain protocols are proposed [12, 13]. Comparing to the other existing blockchain protocols, the Tendermint blockchain has unique characteristics including cross chain transactions system along with in-chain transactions. A new set of validators is selected dynamically for each new block. Recently many studies have analyzed the correctness and fairness of Tendermint blockchain protocol [14]. Identifying many bugs in the code of original Tendermint protocol, this study has proposed fixes for these errors and for achieving correctness and fairness, many enhancements are also proposed. Another recent study [15] has also analyzed the original Tendermint protocol and highlighted various challenges caused by adversary and system model. In this paper, we have also reviewed the contribution in [16] and after further investigation, we propose an enhancement, discussing the design and the Tendermint blockchain protocol in more details. We have made the following enhancements in the original Tendermint blockchain protocol:

- The proposal and voter phases are considered concurrent while the commit phase is considered sequential.
- The voters set is a subset of blockchain nodes. A new methodology is used to decide the size of this subset, investigating the block sensitivity and nodes trustworthiness.
- For the selection of voters set nodes, the random walk algorithm is employed.
- The wait-freedom property is maintained by using a timeout on the voting phase. Thus, all blocks are eventually committed or aborted, which means no one stays in the execution forever and no one waits forever to execute.
- The reasons for voting conflicts and the weakness of consensuses when it is used as a correctness property are investigated using $Round2$ of the algorithm.
- Finally, Detection of Byzantine and failure nodes in the blockchain is shown.

The remainder of this paper is organized as follows: section 2 discuss the related work. Section 3 defines the System Model. Section 4 discusses the proposed modified algorithm. Sections 5 discusses the various concepts. Finally, section 6 concludes the paper.

2. RELATED WORK

The Tendermint is a PBFT consensus-based blockchian protocol [12–14]. Recently, few research studies have identified some issues and proposed enhancements in original Tendermint blockchian protocol [14–16]. Y. Amoussou-Guenou et al. [14] have studied the correctness and fairness of Tendermint blockchain protocol. They have identified issues and many bugs in original protocol. To achieve the correctness and fairness, they have proposed enhancement by fixing the identified issues and bugs. In [15] the original

Tendermint protocol is analyzed. The authors have highlighted various challenges caused by adversary and system model.

Our proposed enhanced Tendermint algorithm is different from the original Tendermint [12, 13] and its enhance variants [14, 15], in the following ways; First, as compared to the existing Tendermint algorithms that are lock based, our algorithm is lock-free. Although, the Tendermint algorithm and its variants use timeout to maintain the correctness. However, it does not provide progressiveness such that in case of transaction failure while holding the lock, it results in an issue with maintaining the progressiveness and the correctness of blockchain. Secondly, we have used the wait-freedom property for eventually commit or abort without staying in the execution forever or wait for execution for the non-deterministic time period. On the other hand, the Tendermint algorithm and its variants use timeout to maintain the correctness. We have proposed novel methods for the selection of the size of validator's subsets and selection of validators in order to achieve fairness. We have also considered the trustworthiness of nodes and sensitivity of data. We have also employed a random walk for the selection of validators whereas, the original Tendermint and its enhanced variants select the validators deterministically which results in unfair reward distribution of the validators.

3. SYSTEM MODEL

This section defines the fundamentals of our blockchain system model. The blockchain [1, 17, 18] consists of a set of nodes (processors) of size n, called P, where $P = \{P_1, ..., P_n\}$; and P_i is any processor belongs to P and i is an integer such that $0 < i < n$. The blockchain contains a distributed ledger L, and each node shares copies of memory objects and L. For example, for a bank blockchain, the memory objects represent the bank accounts, while the operations on the bank account recorded in L.

The operations on the memory objects are either read or write. The read operation returns the data of the memory object, while the write operation updates the data of memory object. A transaction is a sequence of read and/or write operations that execute in isolation and at the end, the transaction commits if it is correct, otherwise, it aborts [19]. For example, if a bank account x has an amount of money equals to 100\$, and a transaction withdraws 10\$, then the new value of x is 90\$, this transaction is correct and commits; however if the result of x is 50\$, then the transaction aborts.

A block B consists of one or more transaction and B commits when all transactions in B commits. On the other hand, B aborts if at least one transaction aborts. Upon the creation of B, it gets the index of proposer as an identifier, namely B_i. It also gets a unique timestamp called ts and it keeps in mind the situation of L; especially the order of the last committed block in L, called Lb_{ts}. Actually, every committed B represents a page in L. The aborted blocks do not show up in L, but for correctness purpose, we consider their ts. Having a blockchain of 4 nodes, Figure 1 shows L of two committed blocks (in the form of $B_{i.ts}$). The blocks are $B_{2.0}$ and $B_{4.2}$. The first block in L is proposed by nodes indexed as 2, and its ts is 0. The node 2 also proposes another block with $ts = 1$ but it is aborted, so it is not recorded in L. The second block in L is proposed by node 4, with $ts = 2$. This means L has blocks with the timestamps of 0 and 2, but it skips the block of $ts = 1$ (because it is aborted).

The blocks are chained in L using hashing, such that each block has the hash of the previous one. Thus, L is unchangeable. The nodes communicate with each other through messages exchange over a network. The blockchain allows broadcasting messages in parallel and a timeout is used to limit the delays. The timeout is specified according to systems specifications. Since P might include a large number of processors, and to reduce the cost of voting processes, we have a subset of validators (voters), called V, where $V \subseteq P$. The size of V (or a number of validators) varies from system to another based on the level of data sensitivity and nodes trustworthiness.

The blockchain may include some network issue, failure nodes or Byzantine ones. A Byzantine node is the one that misleads the system or even lies intentionally [20, 21]. Network issue, failure nodes or no show votes are considered as negative votes. The decision on block B is taken through consensus, which finds the decision of the majority. The majority could be any percentage between 51% and 99%, and that is decided based on the system sensitivity [22, 23]. This helps to reduce the influence of Byzantine nodes. In our algorithm, we test node twice to investigate if it is a Byzantine or not, and to take decision such as excluding it. In our algorithm, we use voters_set[] to represent V. The node P_i broadcasts a proposal of B_i with a unique ts. The nodes in V compute the transactions of B_i and vote to commit or abort B_i. Then, they broadcast the votes in the form of 0 or 1. The received votes are stored in votes[].

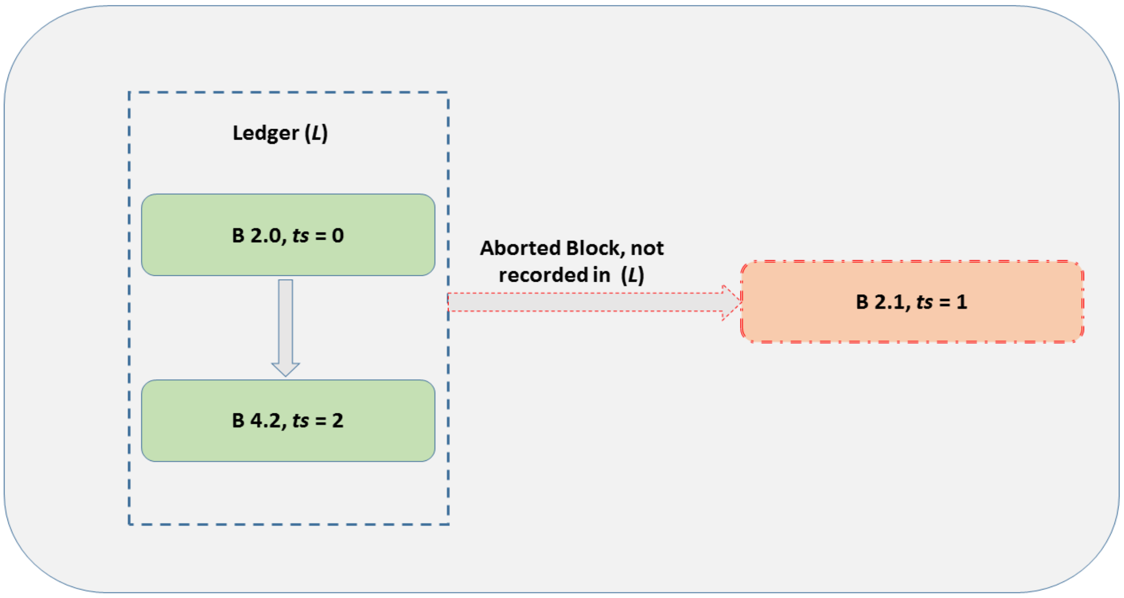


Figure 1. The Ledger L Including the Committed Block $B_{2.0}$ and $B_{4.2}$ Only, While the Timestamps of the Blocks are 0 and 2. The Timestamp 1 is a Unique Timestamp for an Aborted Transaction $B_{2.1}$.

4. PROPOSED ALGORITHM

In the beginning, blockchain algorithm consists of three phases (as shown in Algorithm 1), which are: (i) proposal phase; (ii) voting phase; (iii) commit phase. First, for the proposal phase, any node P_i can propose and create a block B_i at any moment. Now, it sets a timeout for this proposal to complete within. Since some blockchains have a large number of nodes, it is very costly to include all of them in the votes. However, based on data sensitivity and system trustworthiness, P_i decides the $size$ of how many nodes should vote through a function called $size\_voters\_set()$ (that will be explained in details later). Then, P_i calls $voters\_set()$ to decides which nodes are going to vote using a **random-walk algorithm** (that will be explained in details later). After that, P_i finishes proposing phase by broadcasting B_i, L, Lb_{ts} and ts to all nodes in the $voters\_set[]$. Since the proposal phase can be executed by more than one node in parallel, P_i uses a unique integer number (ts) to show the position of B_i in L if B_i commits; otherwise, when B_i aborts we skip its ts. The selection of a unique number ts can be executed in a decentralized manner through many algorithms such as Counting Network [24, 25]. This step ends the proposing phase.

Second, at the beginning of the voting phase, every node in $voters\_set[]$ validates B_i and votes using a function called $vote()$. In $vote()$, the node records 1 if B_i is valid or 0 if it is not. Actually, we create an $arrayvotes[]$ in corresponding to $voters\_set[]$. Next, we wait for some time until all nodes in $voters\_set[]$ vote or the timeout finishes. After that, if timeout finishes while some votes are still missing, then consider all missing votes as votes against B_i using a function called $no\_show\_votes()$. Indeed, for those missing votes, we record 0 in $votes[]$. To end the vote phase, P_i calls $mj\_vote\_Ok()$, checks the votes of the majority, and broadcasts commit or abort message.

Third, the commit phase is the critical section of our algorithm, where we should maintain the $ts's$ of committed blocks in L. It is clear that using timeout all blocks are eventually committed or aborted. For aborted ones we do nothing. However, for committed ones, all nodes must connect them to L in the same order; otherwise, nodes would have different ledgers. In the commit phase, if $B_{i.ts}$ comes directly after Lb_{ts} in L, then B_i commits. For example, if the ts of the last block in the ledger $Lb_{ts} = 5$, and $B_{i.ts} = 6$, it commits. On the other hand, if there is a gap between $B_{i.ts}$ and Lb_{ts}, then B_i has to wait for all blocks in-between to either commit or abort, then B_i commits. For example, if the ts of the last block in the ledger $Lb_{ts} = 5$, and $B_{i.ts} = 8$, then B_i waits for commit and abort broadcasted messages that tells about the blocks of timestamps 6 and 7. Actually, if it receives an abort messages for any of them; it does nothing; while receiving a commit message for any of them means it has to connect it to L before B_i.

As mentioned before, the consensus decision means some nodes may vote against the majority. This happens because of many reasons, such as (i) nodes have computation and processing issue; (ii) being a Byzantine node and lay intentionally; (iii) having an issue with memory consistency [26]. For example, having a memory block x, that equals to 50, when a proposed transaction adds 100 to x, it becomes 150. Since blockchain is decentralized, there is a copy of x in all nodes. If there is an issue of updating x in some parts of the system, then some nodes may have $x \neq 50$ (memory is not consistent), and by adding 100 to x, the result would not be 150, so it votes against the transaction.

```
Algorithm 1: Blockchain Algorithm
// Initialization:
  $L$; //The current ledger
  $B$; //The current block
  $Lb_{ts} \leftarrow 0$; //The timestamp of last block in L
  $ts = 0$; //It is a unique number shows position of $B_i$
  in $L$ if it is committed
  $timeout$; //Waiting timeout for votes
  $size$; //How many nodes voting on $B_i$
  $size2$; //How many nodes voting in $Round2$
  $voters_set[]$; //Who votes on $B_i$ (validators)
  $voters_set2[]$; //Who votes on $B_i$ in $Round2$
  $votes$; //Count the number of votes on $B_i$
  $votes2$; //Count the number of votes in $Round2$
  $votes[]$; //Shows the votes
  $votes2[]$; //Shows the votes in $Round2$

  $Round1$:
   Propose phase:
   //For any processor $P_i$ that proposes new block $B_i$
   $B_i = P_i.create\_new\_block()$;
  $T = current_time + timeout$;
  $size = P_i.size\_voters\_set()$;
  $P_i.voters\_set(size)$; //Decide the voters
   $P_i$.broadcast $B_i$ $(L, Lb.ts, ts) => voters\_set[]$;

  Voting phase:
   //$P_i$ also validates $B_i$ and votes
   if ($P_i.valid(B_i)$) then
     $P_i$.vote(1);
  else
     $P_i$.vote(0);
  //Wait some time for the voters
   Activate receive_votes($votes[]$);
  while (($current\_time < T$) && ($votes < size$)) do
     wait();
  if (($votes < size$)) then
     //Consider all no show votes as zeros
      $no\_show\_votes()$;
  //Checks the majority of votes to commit or abort $B_i$
   if (!$mj\_vote\_Ok(votes[])$) then
     $P_i$.broadcast (abort($B_i$)) ;
     abort($B_i$);
  else
     $P_i$.broadcast (commit($B_i$)) ;
     //Continue with commit phase

  Commit phase:
   // $B_i$ waits until all blocks ordered between $Lb_{ts}$ and
   $B_{i.ts}$ either commit or abort
   while ($B_{i.ts} - Lb_{ts}$ != 1 ) && (all blocks between
   $Lb_{ts}$ and $B_{i.ts}$ are not finished) do
     wait();
  commit();
```

```
Algorithm 2: $Round2$
  //For those nodes that votes against the majority
   $P_i.voters\_set2[]$; // Decide the voting against set
   //$P_i$ selects one of those that voted with the majority
   ($P_j$) and let $P_j$ sends copies of $B_i$ access_set to
   those which voted against
   $P_i$ selects $P_j$;
  $P_j$ broadcasts $B$.(access_set) => $voters\_set2[]$;
  Activate receive_votes($voters2[]$);
  //If $P_i$ in $voters\_set2[]$, it updates its memory,
   validates $B_i$ and votes
   if ($P_i$ in $voters\_set2[]$) then
     Update $B_i$.access_set based on $P_j$;
     if ($p_i.valid(B_i)$) then
       $P_i$.vote(1);
     else
       $P_i$.vote(0);
  $T = current\_time + timeout2$;
  while (($current\_time < T$) && ($votes2 < size2$))
   do
     wait();
  for all $p$ in $voters[]$) do
     //consider all no show votes as failed nodes
      if ($p.vote = NULL$) then
       $no\_show\_votes\_failed()$;
     else
       //if $p$.vote $! = mj\_vote$
        $P.byzantine()$;
```

In some cases, some nodes do not vote. This happens because of many reasons, such as nodes are being in failure; or having communication and network issues. Our algorithm introduces $Round2$ (as shown in Algorithm 2), to investigate these issues. This helps the system to be aware of each node in the blockchain, find network issue, improve the consensus ratio, and improve the system trustworthiness. In the beginning, the proposer node P_i creates a new voters set $voters\_set2[]$ includes all nodes that voted against in $Round1$. P_i selects one node P_j from those nodes that voted with the majority in $Round1$ (P_j could be P_i). Then, P_j broadcasts copies of memory blocks that are accessed by B_i (called B_i. access_set) to $voters\_set2[]$. After that, nodes in $voters\_set2[]$, updates their memory, re-validates B_i and votes again; If P_i belongs to $voters\_set2[]$, it does the same thing. It waits for some time to get the votes. When the $timeout$ finishes, there are three possibilities: (i) some nodes vote (with a majority of $Round1$) correctly, which means they had a memory consistency issue and by updating their memories they get the same results of the majority; (ii) some nodes do not vote, which means they have a failure or communication and network issue ($no\_show\_votes\_failed()$); (iii) If nodes still voting against the majority of $Round1$, then they are Byzantine.

4.1. Correctness proof

Our algorithm is a lock-free algorithm where nodes are able to propose and vote in parallel. However, block committing has to modify the ledger, so it should be serialized and ordered according to timestamps. In the correctness analysis, we show that our algorithm satisfies the Linearizability which is a correctness property that validates concurrent execution. In linearizability, all blocks are ordered in a way that matches a correct sequential execution [27, 28]. Let H be a parallel execution history of all blocks. Now let S be a sequential history, which shows a serialization of the blocks in H based on their timestamps. Indeed, for any two blocks B_i and B_j, if $i < j$, then $B_i <_s B_j$. Note that the timestamp for every block is unique.

Lemma 4..1. *S preserves the real time order of H.*

Proof. According to Algorithm 1, for a block B_i the timestamp i is a unique one, since it is obtained through the Counting Networks. If $B_i <_H B_j$ then, $i < j$. Since S orders blocks in the timestamp order, then we also have that $B_i <_S B_j$, as needed. □

Lemma 4..2. *Our algorithm is wait-free.*

Proof. It is trivial to prove that our algorithm is wait-free, as according to the propose phase in Algorithm 1, every block B_i has a $timeout$. Clearly when the time equals to T, B_i must finish.

After that, the voting phase continues until all votes arrive, or the $timeout$ finishes (means the time reaches T). In such case, the missing votes are considered as negative votes. Then, we check the majority of votes to commit or abort. Therefore, every block eventually finishes within a specific timeout; and the next block eventually executes. □

Lemma 4..3. *The blocks in H are ordered based on their timestamps.*

Proof. Let a block B_i has a timestamp $ts = i$, and the timestamp of the last block in the ledger $Lb_{ts} = x$. where $i > x$.

i. If B_i passes proposing and voting phases successfully, and it is about to commit. If $i - x = 1$ (which means i is ordered directly after x), then B_i commits and connected to the ledger next to B_x.

ii. If B_i passes proposing and voting phases successfully, and it is about to commit. If $i - x \neq 1$, which means there is at least one block with a timestamp between x and i, such that $\{B_{x+1}, ..., B_{i-1}\}$ (where we could have $B_{x+1} = B_{i-1}$) . For simplicity let say there is only one block B_j between B_x and B_i, where $x < j < i$. Then B_i waits until either B_j aborts (receiving abort messages); or B_j commits and becomes the last block in the ledger ($Lb_{ts} = j$). Thus, B_i commits since $i - j = 1$.

    Seeking a contradiction, assume that B_i commits before B_j finishes (commits/abort). Then based on the commit phase in Algorithm1; either $i - x = 1$, which is impossible since $x < j < i$; or B_i receives abort messages of B_j, which is also impossible since the B_j abortion massage means that B_j already finishes, **contradiction**.

    In general case, B_i waits until either some/all of blocks $\{B_{x+1}, ..., B_{i-1}\}$ abort or commit. Actually, B_i knows aborted blocks by receiving abort messages; and knows committed blocks by following the changes of Lb_{ts}. Therefore the ledger preserves the timestamp order.

iii. If B_i fails in proposing or voting phases, Then either the majority vote to abort it, and the abortion massages get sent; otherwise the timeout finishes and the missing votes are considered as negative ones, so it also seems like the majority vote to abort it. Actually, in both cases the aborted block B_i can be ordered easily in the right position in H, since it does not modify the ledger or any local memory.

Therefore, in all cases H preserves the order of S, considering all blocks in H, and from Lemmas 4..1, 4..2 and 4..3, we obtain the following theorem. □

Theorem 4..4. *Any execution history H using our algorithm is linearizable (correct) as it respects the timestamps order that appears in S.*

4.2. Selection of the validator's group size (variable set of validators)

The selection of validators (voters/ miners) in the consensus process is a very important part. After the 51% miner's validation, the new blocks can be added to the chain. Normally in the block verification process, all miners participate in the consensus process. Selecting all miners is a costly process in terms of processing and energy. But sometimes depending upon the sensitivity of data and trustworthiness of miners, it is possible to select a subset of miners out of all the set of miners. We have proposed a new mechanism for the selection of validators. The subset of validators is selected based on the sensitivity of data and the trustworthiness of validators. If the data is highly sensitive and trustworthiness level of the system is low then we need high numbers of validators. On the other hand, if the data sensitivity requirements are low and the trustworthiness high then we need a relatively small number of validators. The following matrix explains (Table 1) the relationship between the three actors (Data sensitivity (di), trustworthiness (ti) and validators (vi)) for the maximum size of the subset of validators.

Table 1. Relationship between data sensitivity, trustworthiness and the size of the subset of validators

Data sensitivity (di)	**Trustworthiness (ti)**	**Validators (vi)**
High	High	50%
High	Low	75%
Low	High	25%
Low	Low	50%

4.3. Selection of validators

In the blockchain, a new block is appended to the chain after the agreement of validators. After we discuss the size of validators set, now we focus one the validator selection. The selection of validators varies according to types of blockchains. For example, in Tendermint blockchain protocol, the subset of validators is selected deterministically using the current history of blockchain. The subset of validators only changes when new blocks are added to the chain. In this paper, we have proposed a novel validator selection mechanism. In the proposed mechanism, we have employed a random walk for the selection of validators. When a new block B is proposed by a proposer, it will select a subset of validators through a random walk. The proposed selection mechanism has few unique features, i.e., the subset of validators are selected non-deterministically and randomly. For every new proposed block, a new subset of validators will be selected. The length of the random walk will be equal to the size of the subset of the validator (explained in the previous section). The proposer node will select a validator randomly from a set of validator and forward the newly created block to the selected validator. The selected validator will select another validator from the list and forward the new block received from the proposer node. The validator selection process will continue until the required numbers of validators have not being selected (selected_validators = = vi). Due to the pure nature of the random walk, we can restrict the random walk to not the selected the already passed validators (i.e., we will exclude them from the group of validators once selected). Figure 2 shows the selection mechanism of validators using random walk.

5. DISCUSSION

5.1. Transaction's validation

Clearly, in blockchain there are two levels of validation and correctness check, which are the validation of transactions and the validation of blocks (of transactions). This section focuses on transactions validation. In fact, when any node creates a new transaction, nodes (miners) picked it up to check the correctness of the transaction, validity, legality and its output [27, 28]. For example, a bank account x belongs to $user1$, and x contains 100$. Now $user1$ withdraws 10$ form x, so the value of x becomes 90$. This is a valid and legal transaction. After that, let $user1$ withdraws 100$, then this is invalid transaction since x does not have sufficient fund to execute the transaction. Also, if x contains 90$ and $user1$ withdraws 20$, but for some reasons x becomes 50$, then it is invalid transaction. This way, nodes check the validity of transactions. If the transaction is valid (correct), it is stored in the node's block, otherwise it is ignored. The node continues the transactions' validation process and place them in its own block until it reaches the block capacity , then it proposes the block to the validators for another validation process. Moreover, it is important to notes that all nodes work in isolation, so each one build its own block regardless the others. This allows different nodes to validate the same transaction, so one transaction may appear in more than one block.

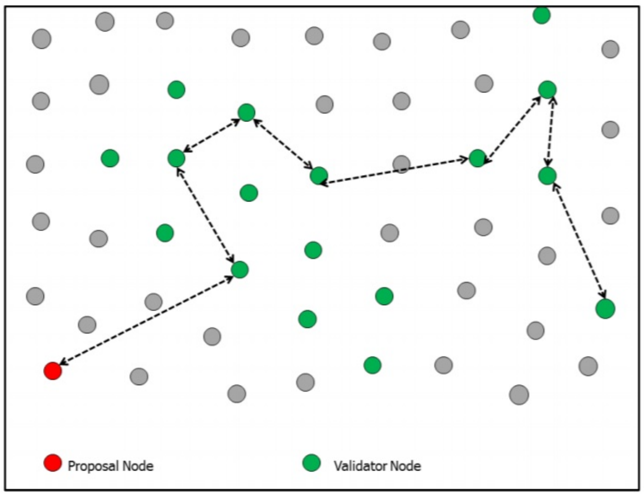


Figure 2. Selection mechanism of validators using random walk

5.2. Block's validation

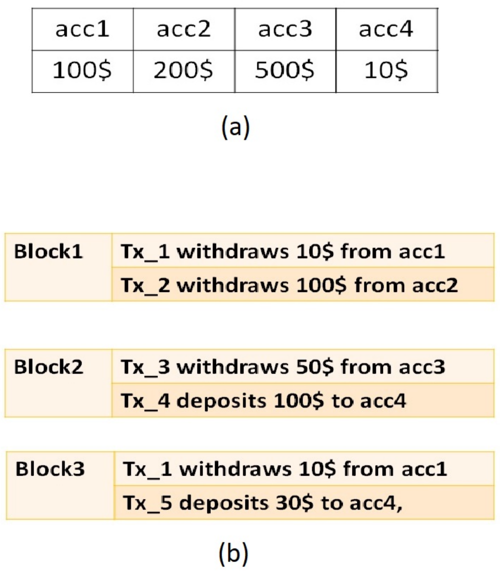
This section focuses on the block validation process. Upon the creation of a new block of transactions, the block should hash it own content (transactions) and produce a fixed size hash number, with specific criteria. This guarantees that no one would change the content of the block, as any change to the block will change the hash number. Blockchain uses PoW to produce the hash number, where the proposer node should solve a mathematical crypto-puzzle [11] (it can use different techniques such as PoS). The hash of the last block in the ledger must be attached as well. The proposer node also includes its digital signature to confirm that the new block is proposed by an authentic and authorized node.

When node proposes a block to the validators, the validators check the block and vote to add it to the ledger (using consensus). Actually the validators validate four items as follows:

- Validators validate the signature of the proposer node to verify that the block is created by a legitimate node.
- Validators validate the PoW to verify that it matches the system criteria and it matches the content of the block (which means the content of the block has not been changed).
- Validators validate the hash of the previous block to fix the order of the block in the ledger.
- Validators validate the content of the block to prevent the conflicts and illegal transactions.

Now, blockchain uses consensus to validate the content of the block to prevent the conflicts and illegal transactions. The consensus decision influences the order of the blocks that are added to the ledger. Figure 3 (a) shows the status of memories for four bank accounts before the block validation and execution. Figure 3 (b) shows three blocks where every block has two transactions. In Block1, Tx_1 withdraws 10$ from $acc1$, so it becomes 90$. Then, Tx_2 withdraws 100$ from $acc2$, so $acc2 = 100$$. In Block2, Tx_3 withdraws 50$ from $acc3$, so $acc3 = 450$$. Then, Tx_4 deposits 100$ to $acc4$, so $acc4 = 110$$. Block3, also has Tx_1 and Tx_5 where it deposits 30$ to $acc4$, so $acc4 = 40$$. Thus, Block3 conflicts with Block1 and Block2. Indeed Block1 and Block3 both include Tx_1, which means if we allows to both of them to commit, then Tx_1 will execute twice and double the withdraw operations on $acc1$, which is not acceptable. Moreover, Block3 conflicts with Block2 since Tx_4 and Tx_5 both of them executes operations on $acc4$, and both of them are not aware of each other. If both of them commit, then the amount of $acc4$ will be either $110$$ or $40$$ (according to the committing order), where the right value is the accumulative of both which is $140$$. Thus, if by consensus, Block1 and Block2 commit first (they can commit in any order as there is no dependency between them), Block3 must be aborted. On the other hand, if by consensus, Block3 commits first, then

Block1 and Block2 must be aborted. The following section focuses on how to order blocks in the ledger.



acc1	acc2	acc3	acc4
100$	200$	500$	10$

(a)

Block1	Tx_1 withdraws 10$ from acc1
	Tx_2 withdraws 100$ from acc2

Block2	Tx_3 withdraws 50$ from acc3
	Tx_4 deposits 100$ to acc4

Block3	Tx_1 withdraws 10$ from acc1
	Tx_5 deposits 30$ to acc4,

(b)

Figure 3. (a) Shows the status of four bank accounts;
(b) Shows three blocks where every block includes two transactions

5.3. Correctness

In this paper, we focus on the correctness of blocks rather than arguing about the correctness of transactions themselves. The correctness of transactions is checked by miners before they propose the blocks. To validate the transactions, nodes may use any approved correctness prosperity such as opacity or Linearizability [27, 28]. Indeed, many researchers restrict the parallelism of blockchain to guarantee correctness [13, 14]. Some algorithms use locks to serialize blocks execution or pipelining them. This increases the execution time and may result in some lock's issues such as deadlock and starvation. However, our algorithm is a lock-free algorithm where nodes are able to propose and vote in parallel. In fact, proposing and voting run in temporary memory and do not require maintaining the ledger. On the other hand, the critical section of our algorithm is when nodes modify the ledger, which is the moment of commit. We propose a timestamped based algorithm where every block has a unique timestamp ts [29]. This ts tells wither the block should commit or wait. Actually, blocks can be aborted directly (based on consensus), since the aborted blocks do not update the ledger. Committed blocks must maintain the ledger with respect to their ts, so every block has to wait for all blocks of timestamps smaller than its own.

5.4. Consensus

The validation of blocks is conducted via consensus. Actually, the consensus is considered as a weak correctness prosperity, it is indeed a fault tolerance property [20, 21, 30]. This means consensus is able to work with some faulty results (votes). For example, the block can commit even with about %49 negative votes. To the best of our knowledge, blockchain algorithms do not consider the weakness of consensus itself. For example, with some financial operations, the decision has to be black or white from all validators, while the consensus allows for some gray area. In our algorithm, we introduce some techniques to support the consensus decision. First, we select the number of validators based on the sensitivity of the data. Second, we decide the consensus percentage based on both data sensitivity and trustworthiness. For example, with our criteria, for some critical matters, the consensus may reach 99%, or abort. Third, we evaluate the trustworthiness of the nodes based on their history of votes. $Round2$ in our algorithm investigates the reasons for having voting conflicts. We investigate the data consistency and we give another chance for the faulty validators to adjust themselves. This improves the data consistency in the blockchain and increases the ratio of consensus as some nodes may be able to vote correctly (with the majority) in the second round. However, for those nodes that do not vote because of failure or network issue. It is important to take some decisions to fix the issues or mark them to be avoided later on. Moreover, they should be treated negatively in the rewards distributions. For Byzantine nodes that insist to mislead the consensus, it is important to classify them under the class of low trustworthy nodes; which decreases their chance to be selected in the voters set and consequently decreases their rewards. Otherwise, blockchain supposes to exclude them. All mention steps are supportive techniques to enhance the consensus decision as a correctness property.

5.5. Wait-freedom

Let us start with a proposed scenario, when one block B_3 with $ts = 3$, about to commit but waits for another block B_2 with $ts = 2$. However, B_2 passes the proposal phase and waits for validators to decide to weather it commits or not. If at least one validator has an issue and does not respond, then B_2, B_3, and all the following blocks would wait for non-deterministic time to commit. To solve such a problem our algorithm maintains the wait-freedom property. Wait-freed means that all blocks are able to complete within a finite number of steps (time) [31]. Indeed, we implement the wait-freedom property using timeout; it starts directly after block creation and finishes within a deterministic time. If it finishes before the block completes, it ignores the pending votes and considers them as negatives. Then takes the decision of the majority to commit or abort. This way blocks can still commit even after the timeout decision. For example, when the voters set contains 10 nodes, 8 of them vote to commit the block, 1 node votes to abort and 1 node delays the vote; then we consider the vote of the one with delay as a negative vote, which makes the ratio 8 to 2; so it commits. Thus, all blocks are eventually committed or aborted, which means no one stays in the execution forever and no one wait for non-deterministic time to execute.

6. CONCLUSION

Blockchain technology has the potential to transform the digitization of industries for more efficiencies. In this paper we analyzed the Tendermint blockchain protocol and proposed various enhancements to improve correctness, fairness, parallelism, performance, progressiveness and trustworthiness.

REFERENCES

[1] S. Nakamoto and A. Bitcoin, "A peer-to-peer electronic cash system," *Bitcoin.–URL: https://bitcoin. org/bitcoin. pdf*, 2008.

[2] R. Cohen, "Global bitcoin computing power now 256 times faster than top 500 supercomputers, combined," *Forbes, November*, 2013.

[3] S. Hakak, W. Z. Khan, G. A. Gilkar, M. Imran, and N. Guizani, "Securing smart cities through blockchain technology: Architecture, requirements, and challenges," *IEEE Network*, vol. 34, no. 1, pp. 8–14, 2020.

[4] N. Herbaut and N. Negru, "A model for collaborative blockchain-based video delivery relying on advanced network services chains," *IEEE Communications Magazine*, vol. 55, no. 9, pp. 70–76, 2017.

[5] Q. Lyu, Y. Qi, X. Zhang, H. Liu, Q. Wang, and N. Zheng, "Sbac: A secure blockchain-based access control framework for information-centric networking," *Journal of Network and Computer Applications*, vol. 149, p. 102444, 2020.

[6] Z. Zheng, S. Xie, H.-N. Dai, W. Chen, X. Chen, J. Weng, and M. Imran, "An overview on smart contracts: Challenges, advances and platforms," *Future Generation Computer Systems*, vol. 105, pp. 475–491, 2020.

[7] W. Khan, M. Rehman, H. Zangoti, M. Afzal, N. Armi, and K. Salah, "Industrial internet of things: Recent advances, enabling technologies and open challenges," *Computers & Electrical Engineering*, vol. 81, p. 106522, 2020.

[8] S. Gao, G. Piao, J. Zhu, X. Ma, and J. Ma, "Trustaccess: A trustworthy secure ciphertext-policy and attribute hiding access control scheme based on blockchain," *IEEE Transactions on Vehicular Technology*, 2020.

[9] S. Moin, A. Karim, Z. Safdar, K. Safdar, E. Ahmed, and M. Imran, "Securing iots in distributed blockchain: Analysis, requirements and open issues," *Future Generation Computer Systems*, vol. 100, pp. 325–343, 2019.

[10] W. Z. Khan, E. Ahmed, S. Hakak, I. Yaqoob, and A. Ahmed, "Edge computing: A survey," *Future Generation Computer Systems*, vol. 97, pp. 219–235, 2019.

[11] I.-C. Lin and T.-C. Liao, "A survey of blockchain security issues and challenges." *IJ Network Security*, vol. 19, no. 5, pp. 653–659, 2017.

[12] E. Buchman, J. Kwon, and Z. Milosevic, "The latest gossip on bft consensus," *arXiv preprint arXiv:1807.04938*, 2018.

[13] J. Kwon and E. Buchman, "Tendermint," 2018.

[14] Y. Amoussou-Guenou, A. Del Pozzo, M. Potop-Butucaru, and S. Tucci-Piergiovanni, "Correctness and fairness of tendermint-core blockchains," *arXiv preprint arXiv:1805.08429*, 2018.

[15] ——, "Dissecting tendermint," in *International Conference on Networked Systems*. Springer, 2019, pp. 166–182.

[16] B. Assiri and W. Z. Khan, "Enhanced and lock-free tendermint blockchain protocol," in *2019 IEEE International Conference on Smart Internet of Things (SmartIoT)*. IEEE, 2019, pp. 220–226.

[17] T.-T. Kuo, H.-E. Kim, and L. Ohno-Machado, "Blockchain distributed ledger technologies for biomedical and health care applications," *Journal of the American Medical Informatics Association*, vol. 24, no. 6, pp. 1211–1220, 2017.

[18] M. Pilkington, "Blockchain technology: principles and applications," in *Research handbook on digital transformations*. Edward Elgar Publishing, 2016.

[19] N. Shavit and D. Touitou, "Software transactional memory," *Distributed Computing*, vol. 10, no. 2, pp. 99–116, 1997.

[20] A. S. Tanenbaum and M. Van Steen, *Distributed systems: principles and paradigms*. Prentice-Hall, 2007.

[21] H. Attiya and J. Welch, *Distributed computing: fundamentals, simulations, and advanced topics*. John Wiley & Sons, 2004, vol. 19.

[22] B. Assiri and C. Busch, "Approximately opaque multi-version permissive transactional memory," in *2016 45th International Conference on Parallel Processing Workshops (ICPPW)*. IEEE, 2016, pp. 393–402.

[23] ——, "Approximate count and queue objects in transactional memory," in *2017 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*. IEEE, 2017, pp. 894–903.

[24] J. Aspnes, M. Herlihy, and N. Shavit, "Counting networks and multi-processor coordination," in *Proceedings of the twenty-third annual ACM symposium on Theory of computing*, 1991, pp. 348–358.

[25] ——, "Counting networks," *Journal of the ACM (JACM)*, vol. 41, no. 5, pp. 1020–1048, 1994.

[26] T. Coppieters, W. De Meuter, and S. Burckhardt, "Serializable eventual consistency: consistency through object method replay," in *Proceedings of the 2nd Workshop on the Principles and Practice of Consistency for Distributed Data*, 2016, pp. 1–3.

[27] R. Guerraoui and M. Kapalka, "On the correctness of transactional memory," in *Proceedings of the 13th ACM SIGPLAN Symposium on Principles and practice of parallel programming*, 2008, pp. 175–184.

[28] M. Herlihy and N. Shavit, *The art of multiprocessor programming*. Morgan Kaufmann, 2011.

[29] F. Cristian and F. Jahanian, "A timestamp-based checkpointing protocol for long-lived distributed computations," in *[1991] Proceedings Tenth Symposium on Reliable Distributed Systems*. IEEE, 1991, pp. 12–20.

[30] G. Tel, *Introduction to distributed algorithms*. Cambridge university press, 2000.

[31] M. Herlihy, "Wait-free synchronization," *ACM Transactions on Programming Languages and Systems (TOPLAS)*, vol. 13, no. 1, pp. 124–149, 1991.