Ain Shams Engineering Journal 12 (2021) 2709–2716

Contents lists available at ScienceDirect

Ain Shams Engineering Journal

journal homepage: www.sciencedirect.com

Electrical Engineering

Enhancing the performance of the blockchain consensus algorithm using multithreading technology

Hossam Samy [a,*], Ashraf Tammam [a], Ahmed Fahmy [a], Bahaa Hasan [b]

[a] Computer Engineering Dept., Arab Academy for Science, Technology and Maritime Transport, Cairo, Egypt
[b] Chairman at ISEC, Chairman at ASC and Founder at Arab Security Conference, Cairo, Egypt

ARTICLE INFO

Article history:
Received 30 October 2020
Accepted 31 January 2021
Available online 27 March 2021

Keywords:
Blockchain
Byzantine fault tolerance
Business
Consensus algorithm
Throughput

ABSTRACT

Blockchain is one of the most powerful and promising technologies nowadays in the IT industry, One of Blockchain's main features is the presence of a Consensus Algorithm, which is responsible for maintaining the security and integrity of the entire blockchain network where all the nodes participating in the network reach a certain agreement. However, some algorithms like the Proof-of-Work require very high energy consumption to reach a single agreement by solving a puzzle and provides a low throughput (3-7 transactions/second), thus, it may not be very reliable in blockchain solutions. In this paper, we aim to provide a reliable choice for different business use-cases by proposing a modification in the Istanbul Byzantine Fault Tolerance voting-based algorithm that provides a higher throughput (up to 1140 tx/s), which will be very important to be used in the use case called a letter of credit which is a part of trade finance (relation between exporter, importer and the bank institutions).

1. Introduction

Blockchain is a decentralized peer-to-peer network that contains a distributed ledger storing blocks of data chained together by a cryptographic hash and shared across nodes of the network that have access to this data. There are main four factors that characterize the blockchain:

1- **Distributed Ledger Technology:** It contains the history of all transactions, append-only with immutable past, distributed and replicated.
2- **Cryptography:** Integrity of the ledger, the authenticity of transactions, privacy of transactions and identity of participants.
3- **Business Logic:** Logic embedded in the ledger, executed together with transactions and from simple "Coins" to self-enforcing "smart contracts".
4- **Consensus Protocol:** It is the procedure to ensure that all nodes in a network apply the same set of transactions to the ledger at each update.

So we can say that the blockchain is a very promising solution for many problems like single point of failure or manipulation of stored data, and in some cases can prevent distributed denial of service (DDoS) attacks. However, implementing a blockchain-based solution for business use-cases requires certain standards to be met, so there are some factors to evaluate the efficiency of a blockchain-based system:

- **Throughput:** Number of transactions per second.
- **Latency:** The time between transaction execution and the block to be committed in the chain.
- **Scalability**: How the network's operations are affected by increasing the number of participants.
- **Security:** Securing the network against cyber-attacks and making it less vulnerable to double-spending and other attacks threatening the system.

Security and Scalability concerns are challenging for current blockchain solutions, as a reason for how blockchain eliminated the need to trust third-party entities to control data flow in different scenarios, and allowed only authorized entities in the network to manage and validate data entering the blockchain. One of the

* Corresponding author.
E-mail address: eng.hossam.samy2018@gmail.com (H. Samy).

Peer review under responsibility of Ain Shams University.

Production and hosting by Elsevier

https://doi.org/10.1016/j.asej.2021.01.019
2090-4479/

biggest security challenges for blockchain solutions as a result of its distributed nature is the *Byzantine Generals Problem* [9] described by Leslie Lamport, Robert Shostak and Marshall Pease in 1982, where a group of Byzantine generals in a commander-lieutenant fashion need to reach an agreement on when they launch an attack on their enemy through exchanging messages. The key concept here is that the army needs to reach consensus about the time of attack depending on the majority of decisions every lieutenant receives in order to win despite the possibilities of having traitors in the army. For an army that consists of **N** number of generals and **F-**number of traitors, the army would still be able to reach a consensus where **N = 3F + 1**, meaning that at least 2/3 of the generals would have to be loyal and at most 1/3 of them are traitors in order not to lose the war. In the world of distributed systems, it is essential to secure the network against the threats it might face, e.g. a process may fail, a device might die, a connection might be interrupted or a node might be malicious. So, a few characteristics for an efficient distributed system have to be present to solve some of the previous mentioned problems:

1- **Crash-Fault Tolerant System:** A system that implements a strategy to deal with cases of having crashing nodes in the network like disconnection faults for example, the crash-fault tolerance algorithm aims to maintain the network's log between online and crashed nodes to preserve the log's state across all nodes. Paxos and Raft are examples of crash-fault tolerance algorithms.
2- **Byzantine-Fault Tolerant System:** A system that implements a strategy to ensure the continuity of system operations while having malicious or bad nodes in the network by limiting their effect on the network as much as possible as long as the majority of the network is honest. PBFT is an example of byzantine-fault-tolerance algorithms.

In this paper, we discuss Blockchain technology from the Byzantine Fault Tolerance overview, how different consensus algorithms tried to secure blockchain's network against faulty nodes, And illustrate how the IBFT algorithm can be optimized to improve its performance. We discuss how to replace the proof of work algorithm with all its problems and delays in transactions with a modified Istanbul Byzantine fault tolerance which is a modified protocol and we will use it in the use case: letter of credit [35] which is a relation between market and banks which will participate to decrease the delay in the transaction and increase the throughput which is very important in the trade finance market. We discuss Blockchain technology from the Byzantine Fault Tolerance overview, how different consensus algorithms tried to secure blockchain's network against faulty nodes, and illustrate how the IBFT algorithm can be optimized to improve its performance. The outline of this paper is divided into six sections. We talk about other related works considering blockchain consensus algorithms in Section 2. Section 3 demonstrates different consensus algorithm approaches to be byzantine-fault tolerant, In Section 4 We showcase our contribution. The steps we took to evaluate the performance of different blockchain platforms like *Hyperledger* and *Quorum* and benchmark results are present in Section 5. And lastly, we state our conclusion and the future work in Section 6.

2. Related work

Achieving that agreement is one of the oldest and biggest challenges in distributed systems providing multiple complications regarding overall system's reliability. Many contributions were made trying to provide a consensus protocol that implements an efficient fault-tolerant mechanism in distributed system environments. For example, PoW and PoS are two examples of proof-based blockchain consensus protocols [19], proof-based protocols rely on proving eligibility for maintaining the distributed ledger's state. In [22] and [33] the authors outline different consensus choices to be considered in public and private blockchains including Proof-of-Work (PoW), Proof-of-Stake (PoS) and Proof-of-Activity (PoW/PoS-hybrid), outlined additional options like: Delegated Proof-of-Stake (DPoS) [24], Proof-of-Burn (PoB) and others. However, no discussion about having byzantine nodes in the network is present. In [21] Nguyen and Kim highlighted voting-based consensus protocols like PBFT and Raft and their performance in comparison to proof-based protocols. Lewis Tseng in issued a survey about the performance of BFT-based protocols with comparison to the fault tolerance techniques implemented in Bitcoin [15]. [29] compares between PBFT and PoA (proof-of-authority) protocols with respect to CAP theorem. Whereas Moniz illustrates the concepts of IBFT (Istanbul Byzantine Fault Tolerance) protocol that is derived from the PoA protocols family and PBFT in [31]. The increased interest in implementing private blockchain-based [23] solutions for fulfilling business needs aided the research done about the most reliable protocol to be used in a private blockchain, where IBFT proved to be a great candidate. Saltini [27] made a correctness analysis of IBFT algorithm to test its liveness, meaning how efficiently the protocol can ensure reaching consensus, with David Hyland-Wood in [30] who built on Saltini's work and performed a liveness and robustness analysis of IBFT to conclude that IBFT protocol does not guarantee Byzantine-fault-tolerance persistence and liveness when operating in an eventually synchronous network. David and Saltini also proposed some modifications for IBFT to ensure optimal persistence for Byzantine-fault tolerance.

3. Blockchain consensus algorithms

A consensus algorithm can be defined as the set of rules that controls the state of the distributed ledger across nodes participating in a blockchain network. Participating nodes must reach an agreement upon these rules altogether in order to append new data to the blockchain while preserving the ledger's state. Next, we demonstrate some of the different consensus algorithms used in blockchain-based systems and how they try to approach consensus between nodes of the network.

3.1. Practical byzantine fault tolerance

Aiming to develop a practical solution for the Byzantine Generals Problem that can be implemented in distributed systems, Miguel Castro and Barbara Liskov introduced a state-machine replication algorithm [2] that they called PBFT (Practical Byzantine Fault Tolerance) in 1999 designed to work in asynchronous network environments with high-performance optimizations. In PBFT, each node in the network has a state-machine replica to maintain overall system state. Besides, nodes are sequenced in a leader–follower fashion where a primary node is called the leader and follower nodes are called backups. To reach consensus, all nodes have to communicate with each other through messages in which certain parameters are set to detect any faulty node behavior. The goal here for honest nodes is to reach an agreement through a majority of votes given that a network of **N** total number of nodes can tolerate **F** faulty nodes where **N = 3F + 1**, which means at least 2/3 of nodes the network must be honest. A Typical view (round) of PBFT has 4 main phases:

1. A client sends a request to the primary node invoking a service operation.

2. The primary node multicasts the request to the backup nodes.

3. Nodes execute the request and send the client a reply.

4. The client waits for **F + 1** replies from different nodes with the same result, where F is the number of possible faulty nodes.

Communication between primary and backup nodes is done using 3-phase protocol as the leader node starts this protocol when it receives a request from a client, nodes exchange messages with each other that contains different parameters like: *current phase*, *view (round) number*, *sequence number of client request*, *sender signature* and other data that help nodes verify other honest participants based on the majority of messages it receives and reject suspicious ones. The three phases are *PRE-PREPARE*, *PREPARE* and *COMMIT*.

1. PRE-PREPARE: The leader assigns a sequence number to the client request, and multicasts a *PRE-PREPARE* message to the backups.

2. PREPARE: If the *PRE-PREPARE* message is valid, the node broadcasts a PREPARE message and enters *PREPARED* state after receiving 2F *PREPARE* messages.

3. COMMIT: If the *PREPARE* message is valid, the node broadcast a *COMMIT* message and enter *COMMITTED* state after receiving F + 1 *PREPARE* messages, upon receiving 2F + 1 PREPARE messages it enters *COMMITTED-LOCAL* state and the client request is executed.

If a client sent a request to the leader and didn't get a reply within a certain amount of time or neither did the backup nodes after *PRE-PREPARE* phase, the client sends the request to all backup nodes where they communicate with the leader to see if the client is served previously or not. Having not responded again, the leader is suspected to be faulty and backup nodes can multicast a *VIEW CHANGE* message where they vote on advancing to a new round and changing the leader.

PBFT is very energy-efficient and gives more room for more transactions, it features the concept of *Block Finality*. Block Finality means when a proposed block is agreed upon in a PBFT network to be added to the chain, that block is final and there is no need to confirm the block again since it already has the approval of the majority of nodes in the network. However, due to its heavy communication mechanisms, it suffers from scalability issues as a huge number of nodes will affect the overall performance of the system drastically. Also, it may fall for sybil attacks as an attacker may try to claim majority by participating with a huge number of malicious nodes.

3.2. Istanbul byzantine fault tolerance

Inspired from a reputation-based algorithm called PoA (Proof-of-Authority) and PBFT [2] consensus algorithms, IBFT (Istanbul Byzantine Fault Tolerance) algorithm was developed in 2017 by *AMIS TECHNOLOGIES* [11] to provide an alternative for PoW and PoS implementations in the *Ethereum* blockchain [32], to be later acquired by *ConsenSys* foundation [3]. IBFT inherits the 3-phase consensus protocol from PBFT where a static validator (leader) proposes a block insertion, in IBFT however, there is a dynamic set of validators from which a proposer is selected in a round-robin fashion. In IBFT, a round starts when a Proposer initiates a block proposal and ends with either a block successful insertion when the majority of validators agree upon addition of block, or a *ROUND CHANGE* which is IBFT's adaptation of *VIEW CHANGE* in PBFT. Round changes happen when block appending fails, a message is invalid or a response timeout occurs. Validator node wait for **2F + 1** valid messages from other validators in order to change its state whether from *PRE-PREPARED* to *PREPARED*, from *PREPARED* to *COMMITED* or from *COMMITTED* to *ROUND CHANGE*. Alongside its *Block Finality* mechanism, *Block Locking* is another feature implemented in IBFT to prevent forking where a majority-agreed block is "locked-in", meaning no other blocks can be considered until the agreed block is attempted to be appended to the chain.

As mentioned above, the set of validators can be changed by removing existing validators or adding new ones, this done using a voting process where a validator is added or removed through a majority vote of **(Floor(N/2) + 1)** that is captured in the block header. IBFT made a few refactors to some fields in *Ethereum's* block header to function properly, we state here some of them:

beneficiary: Identifies the node for which a vote is being cast.

nonce: Specifies the vote direction, contains two values either *AUTH* or *DROP*.

mixHash: A fixed magic number, identifying this block as an IBFT validated one.

extraData: Contains IBFT specific data like the addresses of validators, *ProposerSeal:* identifier for a Proposer , *CommittingSeals:* list of the validators which reported *COMMIT* on this block.

Achieving Byzantine-fault tolerance in the trustless system, like bitcoin, comes with some pitfalls, like PoW huge energy-consumption burdens or PoS Stake-based restrictions, that is not the case with IBFT which is very energy-efficient compared to PoW and cost-efficient compared to PoS, IBFT algorithm is very beneficial when it comes to implementing private and consortium blockchains. An Environment where all validators can be trusted, accountability for actions is present and a reliable transactions throughput can be achieved fulfilling business needs while tolerating byzantine faults and securing system operability.

4. Our contribution

In this section, we propose a modification for IBFT consensus protocol in order to increase transaction's throughput and reduce the time required for execution of requests in a private blockchain. We can summarize the changes made as follows:

- The processes controlling Block Proposal Verification are executed concurrently instead of running chronologically.
- Duration of message timeout between each phase (PRE-PREPARE, PREPARE and COMMIT) is decreased.

When a proposer gets selected, he sends a Block Proposal to the rest of validators triggering the PRE-PREPARE phase. Each validator has to verify the block proposal received to be able to proceed for the PREPARE phase by entering the state of *PRE-PREPARED.* The figure below demonstrates how the processes involved in block verification can be executed concurrently by adding three threads, each thread is responsible for executing different blocks of instructions, ensuring less time required for block validation. Algorithm 1 below details the modified algorithm steps required to complete

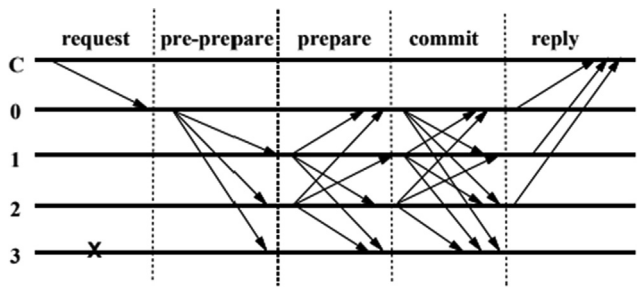


Fig. 1. Communication flow in PBFT's 3-phase Protocol. C is a client. 0 is the leader, and 3 is a faulty backup.

block verification and enter PRE-PREPARED state, the three previously mentioned threads are highlighted in a blue background color.

- **EDIT_THREAD_1:** Handles the verification of the proposer's identity and the set of validators required to communicate with.
- **EDIT_THREAD_2:** Handles the verification of the proposal such that certain parameters must be checked (i.e sequence number and round number) to ensure proposal's validity.
- **EDIT_THREAD_3:** Handles the verification of the block itself, ensuring block finality for validated blocks.

and is broadcasted in case of round timeout. Validator enters PRE-PREPARED state only after completing block proposal verification processes which depends of evaluating the following:

- A block proposal is from a valid proposer, such *proposer* $\in$ *validatorsSet*.
- A block proposal should have the same sequence *currentSeq* and round *currentRound number* as the validator's state.
- A block header contains a valid proposer and validator's signature, valid *hash* representing a number identifying an IBFT block and valid *diff* value of a fixed number.

Algorithm 1: Pseudocode for the proposed modification in IBFT Algorithm

```
constants
        proposer
        validatorSet
variables
        state
        currentRound
        currentSeq
        requestInfo
        msg
        t               // timer
        n               //  nonce: block header parameter
        hash    // mixHash: block header parameter
        diff            // difficulty: block header parameter

function SEND_PREPREPARE(proposer)
 if proposer = currentRound.proposer then
   broadcast  msg <PRE-PREPARE, currentRound,currentSeq, requestInfo>
   start timer
when t expires at t = currentRound.timeout do
   broadcast msg
  <ROUNDCHANGE, currentRound, currentSeq, currentRound+1, requestInfo>
when msg.isReceived = true do
 function EDIT_THREAD_1()                                  <- Modification starts here
   if proposer ∈ validatorsSet([v1, … ,vn-1])
     ^ msg.Sender = currentRound.proposer
   then proposer.isValid ← true
   else return error
 function EDIT_THREAD_2()
   if currentRound = msg.currentRound ^ currentSeq = msg.currentSeq
   then msg.isSynced ← true
   else return error
 function EDIT_THREAD_3()
   if n = blockHeader.nonce ^ hash = blockHeader.mixHash
          ^ diff = blockHeader.difficulty
   then  blockHeader.isValid ← true
   else  return error                                      <- Modification ends here
function ACCEPT_PREPREPARE()
  state.currentState = pre-prepared
  broadcast msg <PREPARE, currentRound, currentSeq, requestInfo>
```

Upon receiving a proposal, a timer *t* is initiated and validator gets to respond with a message denoted as < *msg_name, currentRound, currentSeq, requestInfo* > where *msg_name* is the current phase *currentRound* is the number of the current round , *currentSeq* is the sequence number of the proposal and *requestInfo* is data describing the request. *ROUNDCHANGE* message contains an extra field of *currentRound + 1* which is the number of the next round

5. Experiments and benchmarks

5.1. Blockchain performance evaluation framework

We used a blockchain evaluation framework called *Hyperledger Caliper* [5] to run our benchmarks, a tool that lets you measure the performance of a blockchain network using some predefined use

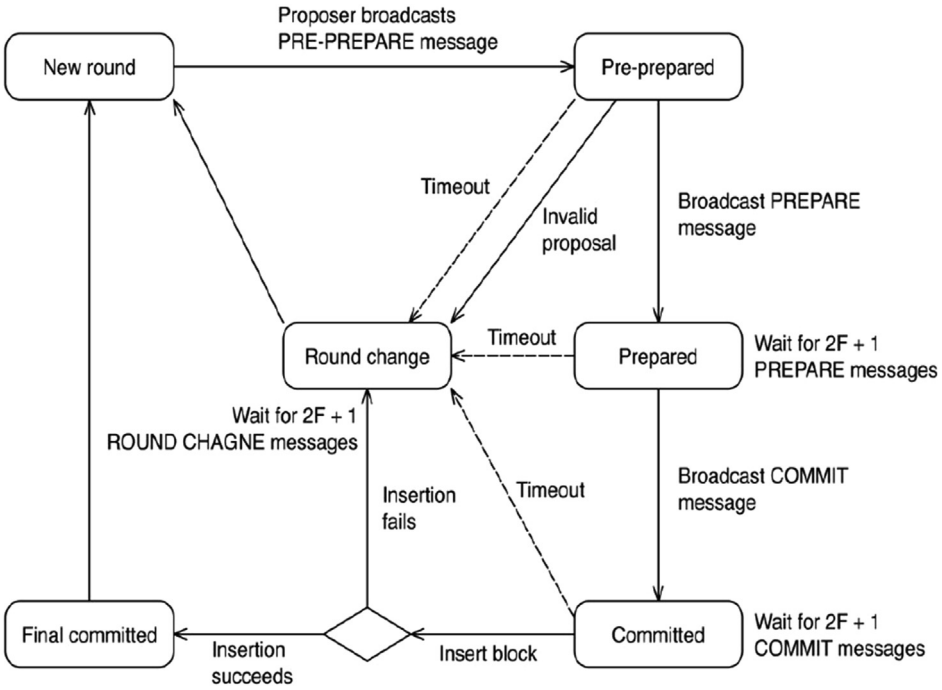


Fig. 2. State Transitions in IBFT algorithm.

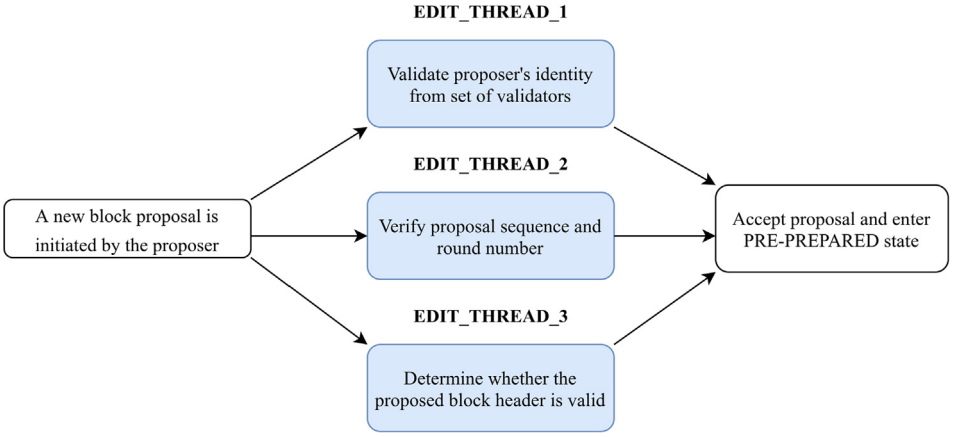


Fig. 3. Overview of the IBFT algorithm modification.

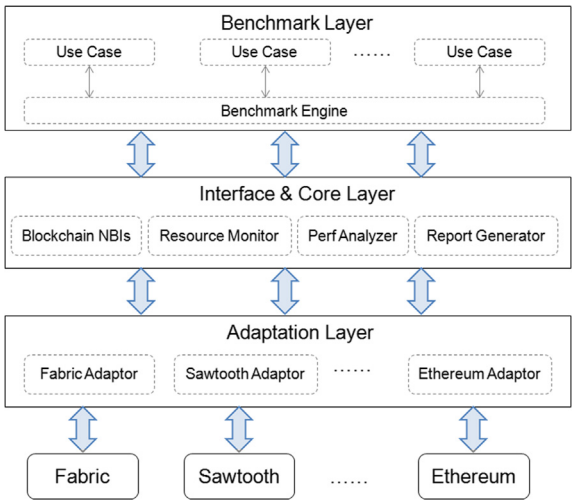


Fig. 4. Hyperledger Caliper Architecture.

cases by generating a workload against the desired system and monitoring its responses in real-time, then generating a report. The Report contains a few metrics that are essential to aid in evaluating network's performance like transactions throughput (Tx/s), transactions latency (secs) and resource consumption. (I.e. CPU usage, Memory usage, etc.). *Caliper* consists of three main layers:

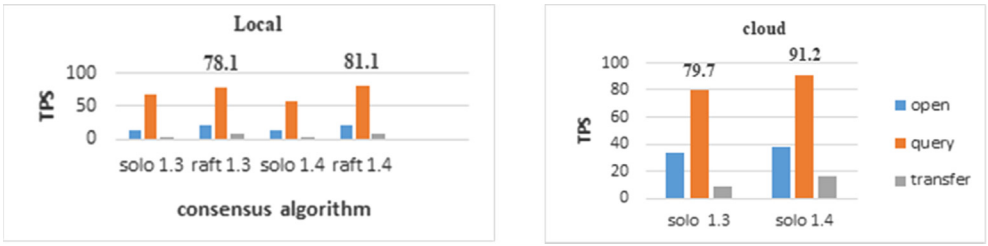
1- **Adaptation Layer:** Responsible for integrating different blockchain implementations into the framework by creating an adaptor for each platform.

2- **Interface and Core Layer:** Responsible for monitoring resources utilization, calculating performance statistics (i.e. tx/s, latency) and generating a report to showcase the data.

3- **Benchmark Layer:** Responsible for implementing case-specific blockchain tests through the benchmark engine where test parameters and conditions can be configured.

5.2. Benchmarking

We used *Hyperledger Caliper* Framework to perform benchmarks on *Hyperledger Fabric* [28] blockchain operating with Solo



Figs. 5 and 6. Solo and Raft transaction throughput in Hyperledger Fabric v1.3 and v1.4 (local and cloud).

Figs. 7 and 8. Solo and Raft transaction latency In Hyperledger Fabric v1.3 and v1.4 (local and cloud).

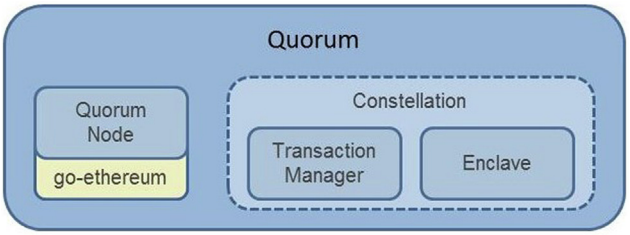


Fig. 9. Basic Quorum Architecture.

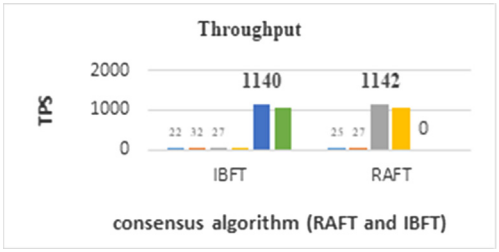


Fig. 10. IBFT and Raft transaction throughput in Permissioned Quorum Networks.

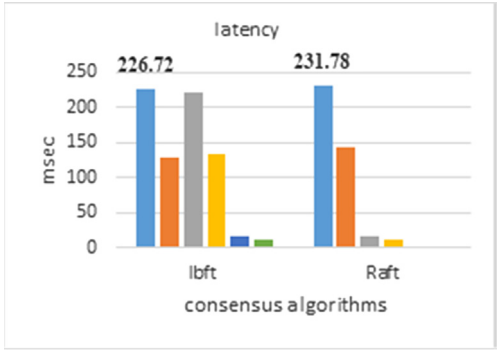


Fig. 11. IBFT and Raft transaction latency in Permissioned Quorum Network.

and Raft ordering consensus protocol, and *Quorum* blockchain operating with Raft consensus and IBFT consensus algorithms. The points of comparison that we are going to focus on are two main metrics: *transaction throughput* which indicates the number of successfully submitted transactions per second, and *transaction latency* which indicates the duration between sending a request and receiving a response. Our development environment is a Google Cloud instance running Ubuntu 18.10 on a Quad-core CPU with 8 GB of memory and SSD storage.

5.2.1. Hyperledger fabric

We compared the performance of Solo and Raft protocols in both *Hyperledger Fabric v1.3* and *Hyperledger Fabric v1.4*. In addition to our cloud Ubuntu instance, we decided to perform benchmarks also on a local machine running Ubuntu 18.10 with 8 GB of memory but less SSD storage capacity. Transaction throughput was nearly the same for Solo and Raft in both local and cloud in *v1.3*, however in *v1.4*, Solo managed to slightly overcome Raft with up to 10–15 more tx/s in the cloud environment. In terms of transaction latency, Solo surprisingly suffers locally from a high latency of 6 s compared to its cloud latency of 2 s in *v1.3* and nearly 5 s locally compared to 1 s in cloud. Whereas Raft in *v1.3* and *v1.4* managed to maintain a latency of nearly 2 s in both local and cloud environments.

5.2.2. Quorum

Quorum [3] is a permissioned *Ethereum-based* Blockchain designed for enterprise applications, it has additional features over *Ethereum* like private transactions, private contract deployments, extended control over participant's identity management and implements an alternative set of consensus protocols [16] like (PoA), Raft and IBFT. We compared the performance of Raft and IBFT protocols in *Quorum* through simulating two different permissioned networks that consists of a total number of 7 nodes, one network operates with Raft protocol and the other operates with a build of our modified IBFT protocol and original IBFT [31] protocol alternatively. For the sake of simplicity, we configured the block time and round timeout in both networks to be 1 s and 5 s respectively. Transactions throughput for original IBFT was hardly reach-

Table 1
Comparison between original IBFT, modified IBFT and Raft consensus algorithms.

Point of Comparison	Original IBFT	Modified IBFT	Raft
Network Type	Permissioned	Permissioned	Permissioned
Private Transactions	Yes	Yes	Yes
Private Smart Contract Deployment	Yes	Yes	Yes
Fault Tolerance	Byzantine-fault **N = 3F + 1**	Byzantine-fault **N = 3F + 1**	Crash-fault **N/2 + 1**
Throughput	low	moderate	moderate
Latency	Moderate	low	low

ing 850 tx/s given a high input rate of 1500 + transactions whereas modified IBFT scored a slightly higher throughput of 1140–1145 tx/s that surprisingly was quite similar to Raft's transaction throughput. Latency-wise, original IBFT reached an average transaction latency of 680 ms compared to almost 227 ms for modified IBFT just below Raft's latency of 232 ms (see Figs. 1–11 and Table 1).

6. Conclusion and future work

In this paper, we addressed the necessity for a consensus algorithm to be fault-tolerant, and proposed a modification to Istanbul Byzantine Fault Tolerance consensus algorithm [10] that aims to solve scalability issues of IBFT in private blockchain systems [1]. We used *Hyperledger Caliper*, a blockchain benchmark tool, to evaluate the performance of open-source platforms like *Quorum* [25] and *Hyperledger Fabric*. *Fabric's* utilization of Solo and Raft protocols as transactions ordering services in *v1.4* produces a reliable throughput and low latency making *Hyperledger Fabric* a good candidate for enterprise blockchain-based solution. On the other hand, *Quorum's* implementation of Raft and IBFT as consensus protocols responsible for maintaining the chain of blocks makes it a more logical option to test our modifications against. Original IBFT roughly made a throughput of 900 tx/s out of more than 1500 transactions input in a network of 7 nodes with latency close to 700 ms. The modified IBFT:

1- Managed to slightly improve the throughput reaching 1140 tx/s and even 1200 tx/s in optimal network conditions.
2- Reduce transactions latency down to 227 ms. Thus, creating a room for a slight improvement in overall permissioned blockchain performance. For future work, we are interested in testing the performance of our modification against other permissioned blockchain platforms and even public blockchain environments. Also, we would like to investigate possibilities of integrating IBFT with other protocols like Raft, considering its potential of huge blockchain performance improvement which combines secure data transmission and finality with scalable and trust-worthy network.

References

[1] Dinh T et al. Untangling Blockchain: A Data Processing View of Blockchain Systems. IEEE Trans Knowl Data Eng 2018;30(07):1366–85.
[2] Miguel Castro, Barbara Liskov, Practical Byzantine Fault Tolerance. 1999.
[3] (Online) Quorum Official GitHub Repository https://github.com/ConsenSys/quorum.
[5] (Online) Hyperledger Caliper, https://hyperledger.github.io/caliper.
[9] Leslie Lamport. Robert Shostak and Marshall Pease, "Byzantine Generals Problem. 1982.
[10] (Online) Istanbul BFT Official Proposal on GitHub https://github.com/ethereum/EIPs/issues/650.
[11] (Online) Article by Zi-Chi Lin from AMIS TECHNOLOGIES introducing IBFT https://medium.com/getamis/istanbul-bft-ibft-c2758b7fe6ff.
[15] Nakamoto, Satoshi. Bitcoin: A Peer-to-Peer Electronic Cash System; 2009.
[16] Chaudhry N, Yousaf MM. Consensus Algorithms in Blockchain: Comparative Analysis, Challenges and Opportunities. In: 2018 12th International Conference on Open Source Systems and Technologies (ICOSST), Lahore, Pakistan. p. 54–63.
[19] Sankar LS, Sindhu M, Sethumadhavan M. Survey of consensus protocols on blockchain applications. In: 2017 4th International Conference on Advanced Computing and Communication Systems (ICACCS), Coimbatore. p. 1–5.
[21] Nguyen Giang-Truong, Kim Kyungbaek. A Survey about Consensus Algorithms Used in Blockchain. J Inform Process Syst 2018;1.
[22] Lucas B, Páez RV. Consensus Algorithm for a Private Blockchain. In: 2019 IEEE 9th International Conference on Electronics Information and Emergency Communication (ICEIEC), Beijing, China. p. 264–71.
[23] Pahlajani S, Kshirsagar A, Pachghare V. Survey on Private Blockchain Consensus Algorithms. In: 2019 1st International Conference on Innovations in Information and Communication Technology (ICIICT), CHENNAI, India. p. 1–6.
[24] Yang F, Zhou W, Wu Q, Long R, Xiong NN, Zhou M. Delegated Proof of Stake With Downgrade: A Secure and Efficient Blockchain Consensus Algorithm With Downgrade Mechanism. IEEE Access 2019;7:118541–55.
[25] Baliga A, Subhod I, Kamat P, Chatterjee S. Performance Evaluation of the Quorum Blockchain Platform; 2018. ArXiv, abs/1809.03421.
[27] Saltini R. IBFT Liveness Analysis. In: 2019 IEEE International Conference on Blockchain, Atlanta, GA, USA, 2019, p. 245–52.
[28] Sukhwani H, Wang N, Trivedi KS, Rindos A. Performance Modeling of Hyperledger Fabric (Permissioned Blockchain Network). In: 2018 IEEE 17th International Symposium on Network Computing and Applications (NCA), Cambridge, MA, 2018, p. 1–8.
[29] Stefano De Angelis, Leonardo Aniello, Roberto Baldoni, et al. PBFT vs Proof-of-Authority: Applying the CAP Theorem to Permissioned Blockchain. 2018.
[30] Roberto Saltini, David Hyland-Wood. Correctness Analysis of Istanbul Byzantine Fault Tolerance. 2019.
[31] Henrique Moniz. The Istanbul BFT Consensus Algorithm. 2020.
[32] (Online) Go-Ethereum, https://geth.ethereum.org/.
[33] Mattila Juri. The Blockchain Phenomenon – The Disruptive Potential of Distributed Consensus Architectures. Berkeley Roundtable on The International Economy (BRIE). Berkeley: University of California; 2017.
[35] Agibalova, Elena N. Ilovaysky, Igor B. Kayl, Yanina Y. Usanova, Viktoria A.. Use of Letter of Credit Form of Payment in the Implementation of Smart Contracts and Blockchain Technology. Scientific and Technical Revolution: Yesterday, Today and Tomorrow 2020:160–70.

Hossam Samy. Born in 1988, He received his B.Sc. in Electrical Engineering, Computer department from Military Technical College (MTC) located in Cairo, Egypt in 2011. In 2016, he became a Senior Cybersecurity Consultant and a Blockchain Team Leader. He's currently joining the Arab Academy for Science, Technology & Maritime Transport (AASTMT) in Cairo to earn a M.Sc. degree from the Department of Computer Engineering. His research interests include blockchain in financial sectors application design, cryptography and consensus algorithm

Ashraf Tammam. He is an Assistant Professor of Computer Engineering at (AASTMT), and Cairo, Egypt. He received his B.Sc. in Electrical Engineering – Computer Engineering from Military Technical College (MTC), Cairo, Egypt in 1994. He received his M.Sc. and PhD in Electrical Engineering - Computer and Systems Engineering from Faculty of Engineering, Ain Shams University, Cairo, Egypt in 2004 and 2011 respectively. He was the Chairman of Information and Decision Support Center (IDSC), Egyptian Cabinet in 2014. His research interest includes Computer and Network Security and IoT

Ahmed Fahmy. He received his Ph.D. in Electronics and Communications in 1981 from Cairo University. He earned a Ph.D. in Computer Engineering from NPS, California USA In 1987. He was the Head of the department of Engineering in the Air Force Academy, Egypt from 1987 to 1989. From 1989 to 1993 he was a Faculty member in Saudi Arabia. From 1993 to the Present he has been in the AAST. During this time, he was the Head of department of the Computer Engineering in both the Qatar and Egypt branches of the AAST until 2015. His research interests are Microprocessor Design, Operating Systems, VHDL Design and Embedded Systems

Bahaa Hasan. Bahaa Eldin M. Hasan earne a B.Sc. and M. Sc. degrees in Electrical Engineering from Faculty of Engineering, Zagazig University in 1978 and 1987 respectively. He received a Ph.D. degree from Ain Shams University under supervision of the Tokyo Institute of Technology in 1994. He founded Arab Security Consultants (ASC) in 2006 and ISEC in 2016. Bahaa is interested in Data Security, Network Security, Computer Security and Ethical Hacking. He earned the Egypt Innovative ICT Award from TIEC in Cairo, Egypt.