FAST: A MapReduce Consensus for High Performance Blockchains

Nida Khan
University of Luxembourg
nida.khan@uni.lu

ABSTRACT

Blockchain platforms when used as a database for IoT systems can resolve data reliability, fault-tolerance, consistency and non-repudiation issues. However, their inherent shortcomings related to their throughput in terms of processed transactions, limit their applicability in such environments in a decentralized way as the underlying network is unable to sustain high workloads. In this paper a fully decentralized high performance consensus mechanism, named FAST, is proposed for a public blockchain. FAST is based on mapreduce paradigm for aggregating and adding transactions on blockchain blocks. FAST was implemented and evaluated in a basic blockchain prototype. A light client for FAST using IPFS, was developed to bring about a reduction in the data stored locally. The obtained results from tests conducted on the prototype depict that FAST exceeds the performance of not just other existing blockchain platforms but comes very close to the throughput of traditional electronic payment networks such as Visa.

CCS CONCEPTS

• **Information systems → Data management systems**; • **Security and privacy → Digital signatures**; • **Computing methodologies → MapReduce algorithms**;

KEYWORDS

Blockchain, Consensus, MapReduce, Round Robin, Lamport's Logical Clocks, Elliptic Curve Cryptography

ACM Reference Format:
Nida Khan. 2018. FAST: A MapReduce Consensus for High Performance Blockchains. In *The 1st Workshop on Blockchain-enabled Networked Sensor Systems (BlockSys'18), November 4, 2018, Shenzhen, China.* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3282278.3282279

1 INTRODUCTION

The Fourth Industrial Revolution is spearheaded by Internet of Things (IoT) and is marked by innovative ways to embed technology within the human society [17]. Another technology at the forefront of this revolution is blockchain, an immutable distributed database, which is heralded as the most disruptive innovation of recent times with the possibility to completely overhaul the existing economic system and social order [15]. IoT is predicted to be the single greatest source of data in the coming two years [9]. This data generated by billions of interconnected, embedded devices and sensors would need a storage medium like a blockchain, that ensures data integrity without intermediaries [8]. However, blockchain is presently plagued by performance issues amongst others, which severely mars it's positioning as a database of choice for processing IoT data workloads [7].

In this paper a novel consensus mechanism is proposed, named FAST, which provides an enhanced throughput with a 100% transaction inclusion probability as opposed to the 70% provided by one of the most popular blockchain platforms, Ethereum [30]. MapReduce forms the foundation of FAST consensus mechanism. A strategy based on round-robin scheduling to choose the **master** node, that will add the next block, is used. Block verification is by means of elliptic curve cryptography and authorship of the block by the **master**. Lamport's clock was used to prevent double-spending and to order the transactions as they are submitted. The work derives inspiration from Ethereum [33], where the limiting factor to a scalable and a high performance blockchain is the *proof of work* consensus. The paper therefore focuses mainly on the components that are different from Ethereum, citing Ethereum sources for the already implemented functionality. The proposed consensus mechanism enhances the throughput from Bitcoin's 7 tps [20] and Ethereum's 25 tps [20] to come close to Visa's 1,667 tps [32]. The high throughput is reached using a completely decentralized network unlike Hyperledger, and without the need to trust intermediaries as in Stellar, which has a throughput of 1000+ tps with 1 billion accounts [20], whereas FAST's 1,333 tps is reached through 5000 accounts.

Section 2 in the paper gives the background on the core concepts used in the proposed FAST consensus. Related work given in 3 gives an overview of the state of the art as well as lays the ground for the motivation behind this work. Section 4 gives an overview of FAST consensus with the implementation and test results discussed in 5 and 6. The conclusion and future work is given in 7.

2 BACKGROUND

Blockchain is a an append-only distributed database in a peer-to-peer environment. It is a series of blocks chained together by means of cryptography. Each block has a hash of the previous block in its header rendering it as an immutable database, which cannot be tampered with easily. The blocks are ordered by timestamps and contain a batch of valid transactions. Everyone can read from the distributed database lending it transparency. Every node willing to append data in a public blockchain gets an economic incentive for adding a new block and hence should have a fair chance to do so. Blockchain removes the need for intermediaries to bring about updating data [24].

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
BlockSys'18, November 4, 2018, Shenzhen, China
© 2018 Copyright held by the owner/author(s). Publication rights licensed to ACM.
ACM ISBN 978-1-4503-6050-0/18/11…$15.00
https://doi.org/10.1145/3282278.3282279

In a distributed decentralized network as in a blockchain there needs to be some mechanism to ensure that only one user updates the blockchain at any instant of time to ensure data consistency. The mechanism to allow secure updating of the state of blockchain is termed as **consensus**. The right to cause this blockchain state transition is distributed among the designated users based on the prioritizing parameter determined by an individual blockchain platform.

MapReduce is a software framework that processes large data sets distributed over several machines [21]. MapReduce involves 3 steps namely *Map*, *Shuffle* and *Reduce*. In the **map** function, each worker node assigns input data according to <*key,value*> pairs. The master node ensures that only one copy of redundant input data is processed. **Shuffle** functions redistributes map outputs to have all the data related to a certain key on one worker node. In *Reduce* step, all the workers are executed in parallel, each of them being in charge of processing output data per key.

Elliptic Curve Cryptography is a form of public-key cryptography that utilizes algorithms based on elliptic curves over finite fields [10]. The private key is an integer while the public key is a point on the elliptic curve. The present implementation employs **secp256k1** [28]) curve with ECDSA **(Elliptic Curve Digital Signature Algorithm)**. Elliptic-curve based systems provide equivalent security against the present known attacks but with much smaller parameters, leading to performance enhancement [16]. This gain in performance is specially relevant for devices in the IoT domain where computing power, memory and battery life is constrained.

Round Robin is a preemptive *First Come, First Served* scheduling. A process is allowed to execute for a certain time interval known as a time-slice or a quantum. If the process is not able to finish its execution within this time-slice or is blocked, then it is preempted and the turn to execute passes to the next process at the *head* of the *run queue*. The preempted process is placed at the back of the run queue for its turn to cycle through the CPU. It is a fair scheduling algorithm as each process gets an equal share of the CPU [12].

Lamport's Logical Clock is used to order events in a distributed system. It is not deemed necessary that the clocks of the different processes are synchronized absolutely. The different processes need not agree what the actual time is but they would agree on the order in which events occur in the distributed system [13].

3 RELATED WORK AND MOTIVATION

Blockchain came into inception with the launch of Bitcoin with the decentralized consensus **proof of work** [24]. Bitcoin currently supports 7 tps. A major limitation to achieving this throughput is the consensus mechanism. Proof of work consumes incredible amounts of energy comparable to Irish national energy consumption [11]. There is an economic incentive attached to update the blockchain, which motivates users to contribute to the consensus. Proof of work also involves computing power and the choice of the one to update the blockchain state and get the economic incentive in return is random with the one in possession of high computing resources having a leading edge. 51% attacks to negate the immutable nature of blockchain can be launched with superior computing resources [19]. The consensus is thus, neither fair nor environmentally viable. The time to add a new block is given as several minutes in Bitcoin [20] and 14 seconds [20] in Ethereum. Thus, there is high latency between transaction submission and it's addition to the blockchain rendering the system misaligned as a payment system. In both the blockchain networks not all submitted transactions are added to the blockchain forcing resubmission on the part of the sender. Utilization of proof of work consensus in blockchains dedicated to IoT systems is thus, an impracticable proposition.

Another category of consensus is **proof of stake**, which involves consensus being dependent on the collateral deposited by a user. The user has an economic incentive in updating the blockchain state and is penalized for bad behavior. Proof of stake is an algorithm that works to favor and possibly enhance economic disparity. It confers the right to update blockchain state only to the economically privileged and suffers from 'nothing at stake' attack. A designated user can update the blockchain state on every fork in the chain to have an economic gain despite the outcome of the fork [18].

There is a third category of consensus, which does not rely on resource ownership for consensus, known as **Byzantine Agreement** [14]. Byzantine agreement is seen in Hyperledger as Practical Byzantine Fault Tolerance [5], in Stellar as Federated Byzantine Agreement [23] and in Tendermint [4] coupled with proof of stake-based membership. Generally the membership in Byzantine Agreement systems is through a central authority. Decentralization is thereby reduced. Scalability is also restricted as seen in Hyperledger where the network fails to function when scaled beyond 16 servers [7]. In Stellar the membership is based on a user's social network, where each user can choose whom he can trust making the consensus decentralized. However the implementation of Stellar involves the use of *anchors*, which brings in the need to trust intermediaries for payments.

The enumeration above lists the major consensus protocols being used together with the most pertinent bottleneck in their integration in large scale networked systems such as IoT. Moreover, it has been shown that Ethereum and Hyperledger have poor performance and scalability when dealing with large-scale data processing workloads [7]. These existing limitations motivated the work on a novel consensus mechanism, FAST, proposed in this paper.

4 FAST OVERVIEW

FAST consensus works in a public blockchain network to resolve any contention amongst the participating nodes to update the state of the blockchain. FAST enables new blocks to be added to the blockchain with the node, who adds the block getting a certain fee from each transaction that is a part of the new block added. MapReduce forms the core of FAST consensus. The *Shuffle* step is omitted as FAST ensures that just one transaction per key (sender account address) exists in one round of consensus. Every node in the network, except those who opt not to participate in the consensus, works both as a **worker** and as a **master** node by turn. Every node as it enters the blockchain network is registered for participation as a light client or a full node. A full node participates in the consensus and every node with the exception of light nodes maintains a FIFO queue of the account addresses of the full nodes.

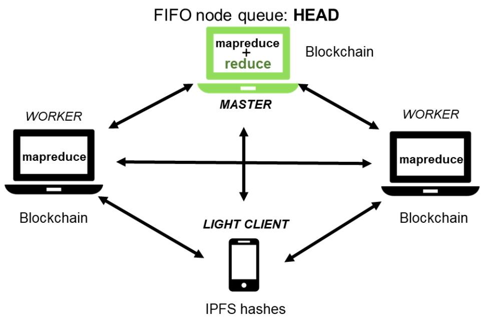

Figure 1: 4-Node Blockchain Network with FAST Consensus

The private-public key pair for a user/ node in the network is generated using ECDSA. The public key is derived from the private key and the account address is a hash of the public key using the same analogy as followed in Ethereum [33]. The node at the front of the queue, the **head**, assumes the responsibility of the **master** node for that particular round of adding a block to the blockchain. Figure 1 depicts an overview of a 4-node blockchain network using FAST consensus. The following outlines one round of adding a new block with FAST consensus:

1. The *worker* node receives the submitted transactions, verifies them and **maps** them to the account address of the transaction sender. If a sender sends two transactions in one round of appending data to the blockchain, then the node retains the transaction with the earliest timestamp. Lamport's logical clocks are employed to order the submission of transactions to ensure that no **double-spending** is taking place.

2. The *master* node aggregates the transactions from all the *worker* nodes and appends the data to its own list of current transactions after **reduce** operation to ensure that no two same transactions are added to the block. Every transaction in the block added to the blockchain has a unique sender account address.

3. The *master* node generates a random token, signs it with it's private key and adds it to the new block with the token. The block also contains a hash of the previous block to ensure immutability as followed in Ethereum and Bitcoin.

4. The *master* node then appends the new block to the blockchain and broadcasts the block to other nodes. It then adds its account address at the back of the FIFO queue. The block is stored locally as well as in *Interplanetary File System* (IPFS) [3]. IPFS is a distributed peer-to-peer file system. Any data submitted to IPFS returns a *hash*, which serves to guarantee data immutability. *IPFS* hash generated is broadcasted so that the light clients can maintain a list of hashes to access the blockchain whenever needed.

5. When a new block arrives the node first compares the account address of the author of the block with the **head** of the queue. If they are the same, the signature verification is correct and the previous hash is the same as the hash of the last block in the blockchain, it appends the arrived block to its local blockchain. Then it pops the FIFO queue to put the node at the front, at the back of the queue.

4.1 Malicious Node

If a malicious worker node attempts to update the blockchain state out of it's turn, then the invalid block will be discovered by other nodes as the account address would not match with the **head** of the FIFO queue. Theft of the private key of the *master* to forge a set of valid transactions from multiple accounts has no incentive as all submitted transactions are added in the new block unlike Ethereum and Bitcoin, where there is a competition to add one's own set of transactions as only the transactions of one winning miner node gets added.

4.2 Fault Tolerance

Every node waits for a certain defined time interval with the timestamp of the last generated block as the base. In case no new block is generated in this time-slice, every node removes the account address at the front of the queue assuming that the node has crashed or is unavailable. Our proposed consensus is aimed for **Crash Fault Tolerant** (CFT) [26] and **Byzantine Fault Tolerant** systems. The usage of Lamport's logical clock-based timestamps take care of *timing failures* while the time-slice takes care of *halting failures*. *Byzantine failures* are discussed in section 7.

4.3 Mutual Exclusion

The distribution of roles in FAST consensus mechanism behaves in a similar way as a distributed mutual exclusion technique, where the critical section is analogous to a node assuming the responsibility of the **master** node. FAST consensus ensures the following properties in the same way as a mutual exclusion technique:

1. Safety: At the most only one node can be the *head* of the queue ensuring that only one node can update the blockchain state at a time.

2. Liveness: A node that is a part of the queue but is not the *head* will eventually get a chance to be the *head* even if the node ahead of it in the FIFO queue crashes or leaves the network through the timeout feature. The timeout ensures that every node in the queue waiting to update the blockchain state gets a chance to do so ensuring the network continues to work despite node failures/ exits/ crashes. Hence there is neither ever a *deadlock* nor *starvation*.

4.4 FAST with *IPFS*

In FAST consensus the block size is variable, as each round of consensus adds all the valid submitted transactions with unique senders to the block. The size can vary from just one transaction in a block to possibly even hundred of thousands.The resulting data size at the higher end can be huge for IoT devices to maintain a local copy of the blockchain database. Work was therefore done on a light client protocol for the FAST consensus. Every node that utilizes the light client stores only the ordered list of *ipfs* hashes to access the blockchain. Only the last block is stored locally. The consensus was tested utilizing even the light client as a participator to update the blockchain state and no implementation issues were discovered but it would be advisable to restrict the light clients from serving as a *master* node on account of limited available memory, which might lead to **omission failures**. The light client might fail to aggregate all the available transactions during the **reduce** operation of the **master** node and no broadcast of the latest block will take place.

5 FAST IMPLEMENTATION

A prototype of FAST consensus was developed and tested on Ubuntu 16.04 LTS using *Python 3.6.5* to build a *Flask 0.12.2* [29] web application for FAST. Ubuntu had a memory of 3.9 GiB, processor was Intel® Core™ i7-6820HQ CPU @ 2.70GHz and OS type was 64-bit. HTTP client, Postman was used for initial testing during development and *cURL 7.47.0* [31], with HTTP and *requests 2.18.4* were utilized for conducting performance, load and stress tests through shell scripts.

The prototype worked by using a dictionary to map the account addresses with the public keys to aid in signature verification as the full functionality for it has been already implemented and tested in Ethereum [6]. Another limitation is that the synchronization of the FIFO queue was implemented using broadcasting. A potential issue with broadcasting can be getting a valid copy of the FIFO queue when a new node enters the network, in the presence of malicious nodes. Hence the prototype would work well for permissioned blockchains only. Usage of **etcd** can resolve this problem and it is discussed further in section 7.

5.1 Block Verification

The verification of a block by the *worker* nodes, broadcasted by the *master* node is by means of verifying the digital signature of the *master* node as well as the previous hash in the block, which should be the same as the hash of the last block in the blockchain.

The source code for block verification function in the FAST consensus is shared in Listing 1. We first get a list of all the node account addresses in the run queue and then reference the first item in the list to get the public key of the head. The signature in the received block is verified with this public key. The hash of the last block is calculated to see if it is the same as the hash given in the *previous hash* parameter of the received block. If both the above conditions are **True**, the block is added to the local copy of the blockchain, otherwise it is considered invalid and discarded.

ECDSA with curve **secp256k1** was used. Secp256k1 is one of the curves recommended by the *National Institute of Standards and Technology* (NIST) [1]. It is used by Bitcoin's public key cryptography [2] and also by Ethereum [33] and thus, has proven blockchain predecessors as a validation of it's utility.

The hash of the block is calculated by using *Secure Hash Algorithm*, SHA-256, which is one of the four hashing algorithms specified by *Secure Hash Standard* [25] and is one of the strongest cryptographic hash functions available. It generates a unique, fixed size 256-bit (32-byte) hash, which serves like an immutable signature for the data in the block. This ensures that the consensus can be used to develop a blockchain, which is secure against tampering and manipulation.

Listing 1: Block Verification by *Workers*

```
def valid_block(self,block):

        token = block['token']
        #get list of node IDs from the queue
        list_nodes=self.q.queue
        temp_url =list_nodes[0]
        #get public key of the queue head
        token_key=self.sorteddict[temp_url]
        #verify the signature in the block
        pub=VerifyingKey.from_string(bytes.
            fromhex(token_key),curve=
            SECP256k1)
        result = self.verify(pub, bytes([
            token]),bytes.fromhex(block['
            signature']))
        #calculate the hash of the last
            block in the chain
        last_blockhash = self.hash_block(
            self.chain[-1])
        #get the previous lash of the last
            block from the received block
        prev_hash=block['previous_hash']
        #check to see if they match
        if result and last_blockhash ==
            prev_hash:
           return 1
        else:
            return 0
```

5.2 IPFS

Python implementation of *IPFS 0.4.17* was used. The code had to be changed to incorporate the use of *ipfsapi* [27]. The FAST consensus nodes were initiated as *ipfs.client* after starting the *ipfs daemon*, which makes use of *Swarm 2.1.6* [22]. Listing 2 shares a few lines of Python code from the function to generate a new block by the *master* node, get the *IPFS hash* for it, broadcast it to the *workers* and add it to the list of IPFS hashes to access the entire blockchain.

Listing 2: IPFS Hash Generation by *Master*

```
def new_block(self, previous_hash):
    #some code before......
    #get the ipfs hash for the block
    ipfshash= api.add_json(block)
    #broadcast the ipfs hash to the worker
        nodes
    self.broadcast_ipfs(ipfshash)
    #add the ipfs hash to an ordered list
    self.ipfs_list.append(ipfshash)
    return block
```

6 FAST PERFORMANCE EVALUATION

A basic blockchain in Python was developed with a few basic functions to support the testing of FAST consensus. Transactions were submitted synchronously from 4 nodes and the latency between the submission of a transaction and the time taken to *reduce* the transaction was observed. It was always found to be one second. The results are depicted in Table 1. It was observed that if the number of nodes is kept constant and the total number of transactions submitted is increased then the performance does not vary a lot. At 6000 transactions, the throughput was 11 tps and at 100000 transactions,

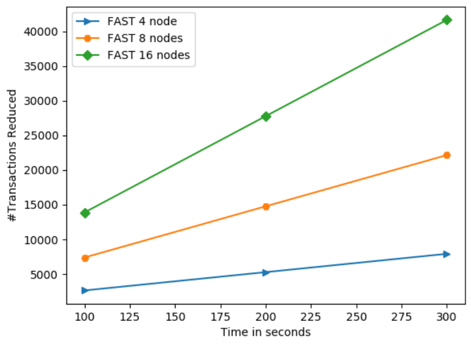

Figure 2: Scalability Test

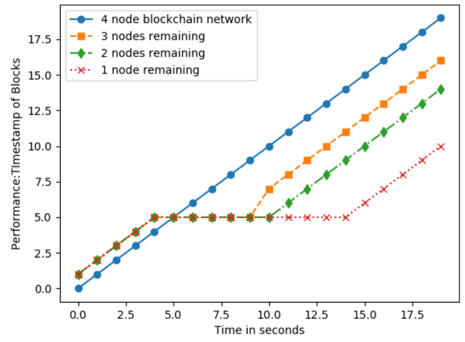

Figure 3: Performance with Node Failures

it was 12 tps. It was also observed that the time taken to submit the transactions on average, is more than the time taken by FAST consensus to *reduce* and add the transactions to the blockchain. The consensus algorithm is fair and each node gets an equal chance to be the **master** node as shown in *#master* (Table 1). The number of blocks generated for the number of transactions submitted is also given. Figure 2 depicts that when synchronous submission of transactions takes place in parallel from 4, 8 and 16 nodes for 5 minutes, the throughput increases with the increase in the number of nodes. **4 nodes record a submission of 7924 transactions**, **8 nodes record 22165** and **16 record 41665 transactions** added to the blockchain. Figure 3 depicts the blockchain performance when one, two and then three nodes crash simultaneously. The timeout in the FAST consensus source code to respond to a node not adding a block has been set to 5 seconds. It was seen that within 6 seconds of a node crash that was at the **head** of the run queue, the network stabilizes to add a new block. 2 node failures result in a recovery time of 6 seconds and 3 in 7 seconds. The basic blockchain prototype that was built to test FAST consensus was exposed to varying number of transactions for a time period of 5 minutes. Changes were made in the source code to accept batches of transactions to simulate transaction submission by x number of users concurrently in a 4-node blockchain network. The tests were performed for x =

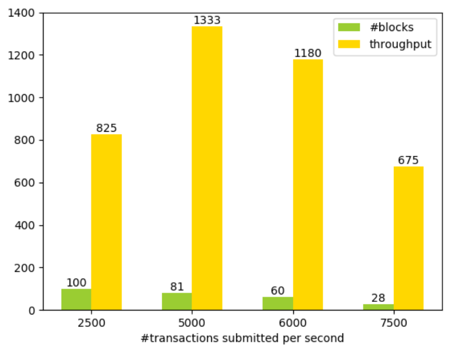

Figure 4: Performance Under Load Tests

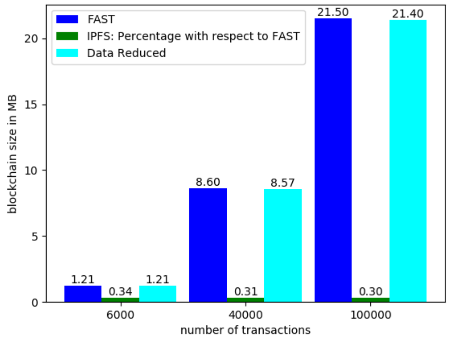

Figure 5: Data Reduction with IPFS

[2500, 5000, 6000, 7500] transactions per second as seen in Figure 4 and it was observed that maximum throughput of **1,333 tps** is obtained with 5000 transactions submitted per second. The subsequent decrease in throughput as x increases is due to increase in load at each of the 4 nodes.

Spike tests were also conducted to see if the consensus can withstand a sudden stress. It was found that on submitting x = *[25000, 40000 and 100000]* transactions and then initiating the function in Listing 2 (**reduce** operation) by the **master** node, all submitted transactions get added in one block in a second. Spike tests simulate parallel submission by x users in a x-node blockchain network.

Tests were conducted with the light client and a massive reduction in the amount of data needed to be stored was observed, making it ideal for use in IoT devices. As shown in Figure 5, it was witnessed that the IPFS hash list to access the blockchain was always 0.3% of the original blockchain size. Thus, with 100000 transactions and a blockchain size of 21.5 MB, the light client needed to store only 65 KB.

7 CONCLUSION AND FUTURE WORK

The paper proposes a novel consensus mechanism, FAST, dedicated to high performance blockchains. An evaluation of it's performance

Table 1: Synchronous Submission of Transactions from 4 nodes

Total Jobs (Transactions)	Submission Time (seconds)	Reduce Time (seconds)	Latency (seconds)	#master node:(1,2,3,4)	#blocks
6000	577	543	1	65, 65, 65, 64	260
40000	5667	5663	1	387, 387, 386, 386	1547
100000	8220	8240	1	894, 894, 894, 894	3577

depicts that FAST works much better than existing consensus protocols being used and comes close to traditional electronic payment networks in performance. The tests performed depict that FAST is scalable and presumably the recorded throughput will be higher in large-scale testing. It was also concluded from the results in Table 1 and Figure 2 that the transaction submission pattern to the FAST mapreduce-based consensus affects the throughput.

FAST is robust to node failures and recovers in a few seconds, which makes it a valid candidate for IoT systems and their associated applications. Additionally FAST can be used to build a payment system, a verifiable data repository and a blockchain-based ID system amongst others. Future work would involve extending the prototype to make use of *etcd* store for maintaining the list of the account addresses of the participating nodes in the blockchain consensus and a successful implementation with *etcd* would make FAST tolerant to *byzantine failures*. *etcd* has been tested in many distributed frameworks like Kubernetes and Docker Swarm. Further, the immutability of the data stored in *etcd* has been proven for distributed networks and so the likelihood of a malicious node tampering with the ordering of the account addresses in the FIFO queue to choose the *master* would be minimized.

ACKNOWLEDGMENTS

The author would like to thank Dr. Radu State, Dr. Abdelkader Lahmadi and Dr. Jérôme François, who guided the initial idea and the paper with their valuable comments and helpful suggestions.

REFERENCES

[1] 1901. National Institute of Standards and Technology. https://www.nist.gov/ U.S Department of Commerce.

[2] 2018. Secp256k1. Retrieved August 15, 2018 from https://en.bitcoin.it/wiki/Secp256k1

[3] Juan Bennet. 2014. IPFS - Content Addressed, Versioned, P2P File System. Retrieved August 15, 2018 from https://github.com/ipfs/reading-list

[4] Ethan Buchman. 2016. *Tendermint: Byzantine Fault Tolerance in the Age of Blockchains.* Master's thesis. The University of Guelph. (In partial fulfillment of requirements for the degree of Master of Applied Science in Engineering Systems and Computing).

[5] Miguel Castro and Barbara Liskov. 1999. Practical Byzantine Fault Tolerance. Proceedings of the Third Symposium on Operating Systems Design and Implementation.

[6] Ethereum JavaScript Community. [n. d.]. ethereumjs/ethereumjs-util. Retrieved August 23, 2018 from https://github.com/ethereumjs/ethereumjs-util/blob/f53d34aec3213c39d117ddabf25dcbddca13ce83/index.js#L351-L367

[7] Tien Tuan Anh Dinh, Ji Wang, Gang Chen, Rui Liu, Beng Chin Ooi, and Kian-Lee Tan. 2017. BLOCKBENCH: A Framework for Analyzing Private Blockchains. In *Proceedings of the 2017 ACM International Conference on Management of Data (SIGMOD'17)*. ACM Digital Library, 1085–1100. https://doi.org/0.1145/3035918.3064033

[8] Bin Liu et al. 2017. Blockchain Based Data Integrity Service Framework for IoT Data. IEEE. https://doi.org/10.1109/ICWS.2017.54

[9] Bret Greenstein. 2018. IoT Trends in 2018: AI, Blockchain, and the Edge. Retrieved August 13, 2018 from https://iot.ieee.org/newsletter/january-2018/iot-trends-in-2018-ai-blockchain-and-the-edge

[10] Darrel Hankerson, Alfred Menezes, and Scott Vanstone. 2004. *Guide to Elliptic Curve Cryptography.* Springer-Verlag, New York.

[11] Gareth Jenkinson. 2017. Bitcoin Mining Uses More Power Than Most African Countries. Retrieved August 13, 2018 from https://cointelegraph.com/news/bitcoin-mining-uses-more-power-than-most-african-countries

[12] Paul Krzyzanowski. 2015. Operating Systems - Process Scheduling/ CS 416. Retrieved August 13, 2018 from https://www.cs.rutgers.edu/~pxk/416/notes/07-scheduling.html

[13] Leslie Lamport. 1978. Time, clocks, and the ordering of events in a distributed system. *ACM Digital Library* 21 (July 1978), 558–565. Issue 7. https://doi.org/10.1145/359545.359563

[14] L. Lamport, R. Shostak, and M. Pease. 1982. The Byzantine Generals Problem, Vol. 4. ACM Transactions on Programming Languages and Systems, 382–401.

[15] Marco Lansiti and Karim R. Lakhani. 2017. The Truth About Blockchain. (January-February 2017). https://hbr.org/2017/01/the-truth-about-blockchain

[16] Kristen Lauter. 2004. The Advantages of Elliptic Curve Cryptography for Wireless Security/ Microsoft Corporation. Retrieved August 13, 2018 from https://www.researchgate.net/profile/Kristin_Lauter/publication/3435958_The_Advantages_of_Elliptic_Curve_Cryptography_for_Wireless_Security/links/0a85e536d732dae938000000.pdf

[17] Guoping Li, Yun Hou, and Aizhi Wu. 2017. Fourth Industrial Revolution: technological drivers, impacts and coping methods. *Springer* 27 (August 2017), 626–637. Issue 4. https://doi.org/10.1007/s11769-017-0890-x

[18] Wenting Li, Sebastien Andreina, Jens-Matthias Bohli, and Ghassan Karame. [n. d.]. Securing Proof-of-Stake Blockchain Protocols. In *Data Privacy Management, Cryptocurrencies and Blockchain Technology*, Vol. 10436. Springer, Cham, 297–315. https://doi.org/10.1007/978-3-319-67816-0_17

[19] Xiaoqi Li, Peng Jiang, Ting Chen, Xiapu Luo, and Qiaoyan Wen. 2017. A Survey on the Security of Blockchain Systems, Future Generation Computer Systems.

[20] Lindsay Lin. 2018. Stellar- Uniqueness/ Differentiation - II. Retrieved August 18, 2018 from https://stellarcommunity.org/t/stellar-meetup-in-singapore/1665/2

[21] Diana MacLean. 2011. A Very Brief Introduction to MapReduce. Retrieved August 12, 2018 from http://hci.stanford.edu/courses/cs448g/a2/files/map_reduce_tutorial.pdf

[22] Frederic Mahe, Torbjorn Rognes, Christopher Quince, Colomban de Vargas, and Micah Dunthorn. 2014. Swarm: robust and fast clustering method for amplicon-based studies. *PeerJ* 2 (9 2014), e593. https://doi.org/10.7717/peerj.593

[23] David Mazieres. 2016. The Stellar Consensus Protocol: A Federated Model for Internet-level Consensus. Retrieved August 13, 2018 from https://www.stellar.org/papers/stellar-consensus-protocol.pdf

[24] Satoshi Nakamoto. 2008. Bitcoin: A Peer-to-Peer Electronic Cash System. Retrieved August 13, 2018 from https://bitcoin.org/bitcoin.pdf

[25] Information Technology Laboratory NIST. 2015. Secure Hash Standard (SHS). https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.180-4.pdf

[26] Diego Ongaro and John Ousterhout. 2014. In Search of an Understandable Consensus Algorithm. USENIX Annual Technical Conference.

[27] py-ipfs-api team. 2017. Python IPFS API Documentation. https://media.readthedocs.org/pdf/python-ipfs-api/latest/python-ipfs-api.pdf

[28] Certicom Research. 2010. SEC 2: Recommended Elliptic Curve Domain Parameters. http://www.secg.org/sec2-v2.pdf

[29] Armin Ronacher. 2017. Flask Documentation. Retrieved August 15, 2018 from https://media.readthedocs.org/pdf/flask/latest/flask.pdf

[30] ETH Gas Station. 2017. Current Dynamics of Transaction Inclusion on Ethereum. Retrieved August 7, 2018 from https://medium.com/@ethgasstation/current-dynamics-of-transaction-inclusion-on-ethereum-ae8912edc960

[31] Daniel Stenberg. 2017. man pages section 1: User Commands. Retrieved August 15, 2018 from https://docs.oracle.com/cd/E86824_01/html/E54763/curl-1.html

[32] Jan Vermeulen. 2016. VisaNet - handling 100,000 transactions per minute. Retrieved August 13, 2018 from https://mybroadband.co.za/news/security/190348-visanet-handling-100000-transactions-per-minute.html

[33] Dr. Gavin Wood. [n. d.]. ETHEREUM: A Secure Decentralized Generalized Transaction Ledger, EIP-150 Revision. Retrieved August 23, 2018 from http://gavwood.com/paper.pdf