Evaluating Blockchains for IoT

Runchao Han
School of Computer Science
The University of Manchester
Manchester, UK
Email: runchao.han@student.manchester.ac.uk

Vincent Gramoli
School of IT
University of Sydney
Sydney, Australia
Email: vincent.gramoli@sydney.edu.au

Xiwei Xu
School of CSE
Data61-CSIRO and UNSW
Sydney, Australia
Email: xiwei.xu@data61.csiro.au

***Abstract*—As proof-of-work blockchains are inherently energy greedy and offer probabilistic guarantees, blockchains based on Byzantine consensus appear as a promising technology to track billions of connected devices. In this paper, we evaluate the performance of prominent blockchains that solve the classic Byzantine consensus problem. Our results show that while offering reasonable throughput their performance usually do not scale to tens of devices and drops dramatically as the number of devices increases. This study motivates the need for solutions that solves the Blockchain consensus problem, a scalable variant of the classic Byzantine consensus problem but dedicated to blockchains.**

***Keywords*—Private blockchains, consortium blockchains, Internet of Things**

I. Introduction

Blockchain is promising but enterprises have been reluctant at adopting proof-of-work blockchains [1] because they offer probabilistic guarantees [2], can be subject to double-spending [3], especially in a consortium or private context [4]. Instead, consortium and private blockchains, which often solve deterministically the classic Byzantine consensus problem, are typically more secure when some assumptions are met. As classic Byzantine consensus protocols [5] were originally designed for small networks, it remains unclear whether one can adapt them to scale to numerous blockchain devices as required by the Internet of Things (IoT).

To address this problem, we evaluate prominent consortium blockchains, including Hyperledger Fabric (HLF) v0.6 with the Practical Byzantine Fault Tolerance (PBFT) consensus protocol [5], HLF v1.0 with the Byzantine Fault-Tolerant State Machine Replication (BFT-SMaRt) consensus protocol [6], and Ripple with the Ripple consensus protocol [7]. Some efforts were recently devoted to evaluate blockchains [8], however, they aim at comparing the performance of inherently slow proof-of-work blockchains to faster blockchains based on Byzantine consensus but we are not aware of experimental evaluation focused on energy-efficient technologies.

Our results show that these blockchains are reasonably fast but their performance degrade significantly with the increase in the number of devices.

II. Related Work

Various blockchains have been proposed for different purposes, but only few proposals aim at targeting the Internet of Things (IoT). Applying blockchains to IoT will likely require to adapt some of the mechanisms of already existing blockchains, especially their consensus implementation.

R3 explored various blockchain technologies, including Ethereum [4], before proposing a distributed ledger, called Corda [9], mainly to process and record transactions between financial institutions. Corda aims at supporting two consensus protocols, Raft [10] and BFT-SMaRt [6]. As the former is insecure, we focused our attention on the latter that tolerates less than $\frac{n}{3}$ malicious behaviours: BFT-SMaRt is a high performance Byzantine Fault Tolerant State Machine Replication that relies on a leader to solve the classic Byzantine consensus problem that predates blockchains. At the time of writing its implementation in Corda was not ready, so we focused on the variant of Hyperledger Fabric that implements BFT-SMaRt as well.

Hyperledger Fabric v0.6 is a customizable consortium blockchain platform [11] that supports smart contracts called "chaincodes". Chaincode supports existing programming languages like Go and Java. Hyperledger Fabric v0.6 relies on PBFT, a Byzantine consensus protocol that was designed 16 years ago for local area network and is still prominent today [5]. As Hyperledger Fabric v1.0 is known to be insecure in the presence of malicious behaviors [12], a fork of v1.0 master branch was changed to invoke a BFT ordering service based on BFT-SMaRt. This branch uses gRPC and supports only timestamp broadcast rather than the message types offered by the main branch but copes with malicious behaviors.

Ripple is a blockchain platform with a strong focus on financial applications. It sets a wide range of transactions for accounts and their XRP currency exchanges, but it lacks the support of smart contracts. It runs its own Ripple consensus algorithm [7]. Although the correctness of the Ripple consensus algorithm has been debated [13], it is out of the scope of this paper to prove whether the code is correct.

IOTA [14] is a cryptocurrency for IoT. As opposed to the classic Byzantine consensus technologies, it does not totally order transactions but builds a directed acyclic graph (DAG) of transactions, called *Tangle*. To commit new transactions, IOTA requires currently a single coordinator, which may become a single point of failure, as the failure of this coordinator

978-1-5386-3662-6/18/$31.00 ©2018 IEEE

prevents transactions from being executed[1]. IOTA is still under development and expected to be decentralised in the future. Commercial products also rely on DAG of transactions [15], unfortunately their code is not available.

The Red Belly Blockchain[2] relies on recent variants of consensus, called Blockchain Consensus [16], especially designed for scalability. Its analysis is part of future work.

III. Evaluating Byzantine Consensus Blockchains

The main focus of this study is the comparison and analysis of mainstream secure blockchains. We selected the three following blockchain platforms:

1) Hyperledger Fabric v0.6 with PBFT consensus [5]. In particular, the reason why we complemented our study with the prototypical HLF version using BFT-SMaRT is because the consensus of HLF v0.6 is known to have unresolved bugs.[3]
2) Ripple with the XRP consensus algorithm [7]: In Ripple, a validator is usually a trusted node that is authenticated by Ripple Inc. in the public network. To speed up the execution, we created our own local validator rather than a public validator.
3) Hyperledger Fabric, based on v1.0, with BFT-SMaRt consensus [6]. Because this branch uses gRPC, which supports only timestamp broadcast rather than the message types offered by the main branch, we had to adapt our benchmark when using BFT-SMaRt.

A. Distributed Experiments

We ran some experiments on a distributed system of 32 machines using Emulab, an environment providing interconnected heterogeneous and physical machines for distributed system experiments. To this end, clients ran Node.js v6.11.4 on Ubuntu 16.04 to generate the load with 2 types of machines equiped with:

- Two 64-bit E5-2630 Haswell processors with 8 cores running at 2.4 GHz, 64 GB 2133 MT/s DDR4 RAM (8 x 8GB modules), one Intel SATA SSD with 200 GB and 2 x 1 TB 7200 rpm SATA disks.
- One 64-bit E5-2630 Nehalem processor with 4 cores running at 2.4 GHz, with 12 GB 1066 MHz DDR2 RAM (6 x 2 GB modules), one Seagate SATA disk with 250 GB and one 500 GB 7200 rpm Western Digital SATA disk.

B. Adapting the Workload to the Blockchain APIs

The difficulty in evaluating different blockchains lies in the lack of interface standards. We constructed a generic workload that performs the same functions on the different blockchains interface. Because, smart contracts are not supported by Ripple, we designed the workload to perform a transfer from an account A to another account B with the amount 1.

TABLE I
Parameters of the workload

Parameter	Description
Method	The HTTP method used for the request
HTTP Headers	HTTP Headers of the request
HTTP Data	HTTP Data of the request(can be a JSON string)

1) The HLF v0.6 API: It provides operations that maintain key-value pairs, also called *states*. Therefore, to create an account, the client can just put a key as the username as well as a corresponding value standing for the balance with an initial value. The transfer of money is represented as an atomic operation that the balance of A decreases and the balance of B increases.

Operations which change the state by the function `PutState()` are non-blocking operations called *invocation*s. Therefore, the client can know whether the transaction has been executed only if it queries the blockchain, which is a blocking operation.

2) The Ripple API: It specifies the format of an account as a hash value, which is created by a passphrase, included in the network with the consent of the root account, and activated by a transfer from another existing account to it. The transfer transaction is called *payment* in Ripple. Therefore, we designed a workload for Ripple as a payment transaction from an account to another account activated in this network with amount 1, which is identical to the transaction in HLF. Similarly to HLF, the Ripple interface is non-blocking in that a transaction commitment is not acknowledged.

3) The HLF with BFT-SMaRt API: It is an experimental branch of the BFT ordering service for HLF. The current version only supports the timestamp broadcast message type, and was tested in order to make a comparison with PBFT ordering in HLF v0.6. As opposed to HLF v0.6, this newer version uses gRPC.

C. Applying a Workload on the Blockchains

All experiments were based on the `loadtest` library of Node.js, which is a tool for testing the performance of a backend service by sending requests. This homogenises the interfaces of the blockchains as both HLF and Ripple supports it through HTTP.

A single HTTP request is defined as a JSON object, which guarantees that the transaction submissions to Ripple and HLF are identical. Again, the transaction invocations are non-blocking. In other words, when a transaction is submitted by HTTP, an HTTP response about the status of the transaction is sent from the server, while the transaction is being processed. The client cannot know if the transaction has committed unless it queries the server again.

The loadtest framework provides the control over the requests sent to the cluster. It offers the possibility to test both platforms by specifying the workload and the scheme of sending workloads. Critical parameters of the workload are listed in Table I, which correspond to characteristics of the HTTP protocol. We increased the workload in terms of

[1] https://cryptovest.com/news/iotas-53-hour-network-standstill-heightens-investors-worst-fears.

[2] http://redbellyblockchain.io.

[3] https://jira.hyperledger.org/browse/FAB-707.

TABLE II
PARAMETERS OF THE EXPERIMENTAL SETTINGS

Parameter	Description
Endpoint	The URL of receiving requests
Concurrency	The level of concurrency
Max Seconds	The duration of sending requests
Workload	The workload definition
Start QPS	The requests per second when the testing starts
QPS Stride	The increase of the QPS to the last QPS
End QPS	The highest QPS of the test
Sleep Period	The period of rest between different QPS testing

TABLE III
PARAMETERS OF THE WORKLOADS FOR HLF V0.6 AND RIPPLE

Parameter	HLF v0.6	Ripple
Method	POST	POST
MIME Type	application/json	application/json
HTTP Data	Invoking chaincode	Payment transaction

request rate step by step and finally gathered related metrics under different request rates. The parameters of a single load testing are listed in Table II.

1) Requests: As for HLF v0.6, to enable a customized transaction we deployed a corresponding chaincode, coded in the high-level programming language Go. The official example includes a chaincode supporting a simple money transfer transaction, which is deployed on the cluster first, then the cluster accepts multiple requests invoking this transaction.

Ripple has a set of well-defined transactions but lacks the support of customized transactions. The defined transactions are related to financial systems, including the transfer. However, accounts are inactive at first. The account address is derived from the public key of it, which is generated by a set of cryptographic steps. To activate an account, another account should transfer an amount of money to its address. The paid address will be activated if the amount reaches the threshold, which is decided by the whole network dynamically. Therefore, to experiment on a private Ripple network two accounts should be activated via the approach above.

However, HLF with BFT-SMaRt ordering only supports the timestamp broadcasting with gRPC rather than other transaction types like the smart contract invocation. Therefore, the testing of BFT-SMaRt ordering was done by its inherited performance testing module. For all the experiments, we use a single orderer. While their role is unclear, changing the number of orderers does not seem to impact performance significantly [12].

2) Testing Steps: Based on the aforementioned principles, the testing consists of multiple automated steps: (1) Setting up the cluster, (2) deploying the chaincode (for HLF), (3) activating accounts (for Ripple), (4) setting parameters of workloads, (5) setting parameters of sending workloads, (6) launching the load testing for 10 times, (7) analyzing the results. Parameters of Step (4) and Step (5) for both platforms are listed in Table III and Table IV, respectively.

TABLE IV
PARAMETERS OF THE EXPERIMENTS FOR HLF V0.6 AND RIPPLE

Parameter	HLF v0.6	Ripple
Endpoint	localhost:7050/chaincode	localhost:5005
Concurrency	10	10
Max Seconds(s)	10	10
Start QPS	200	500
QPS Stride	200	500
End QPS	4000	5000
Sleep Period(s)	20	20

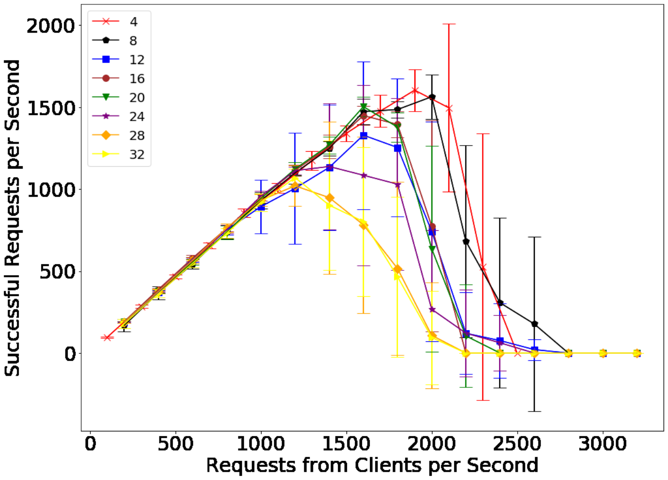

Fig. 1. **The throughput of a node with the increase of the request rate in HLF v0.6.** The QPS first increases but then drops sharply, finally becomes 0.

IV. RESULTS AND ANALYSIS

The load was applied over $4, 8, 12, 16, 20, 24, 28$ and 32 independent physical machines. In this section, we describe the observed results and we analyze the performances of the chosen platforms. HLF with BFT-SMaRt consensus is evaluated separately because of its differences.

A. Metrics of the Load Testing

1) Query per Second: The number of queries per second (QPS) is a common indicator of throughput of a system. It translates into the number of requests the server can treat within a second [17]. The request usually refers to the HTTP request. Obviously, the more requests the system processes in a second, the better it performs.

2) Latency: The latency is the system delay. In other words, it means the delay of a request or a task which the system processes [17]. A system usually processes multiple requests, so the latency is usually the average value of latencies in a period. As the invocations are non-blocking, we did not compute the latency as the time between a transaction is invoked and its commit is acknowledged.

B. Blockchain Throughput as the Request Rate Increases

The result of HLF v0.6 is shown in Fig. 1 whereas the result of Ripple is shown in Fig 2. In both figures, we observe that for each cluster, the throughput in either system increases with the growth of the request rate first, and then reaches a peak.

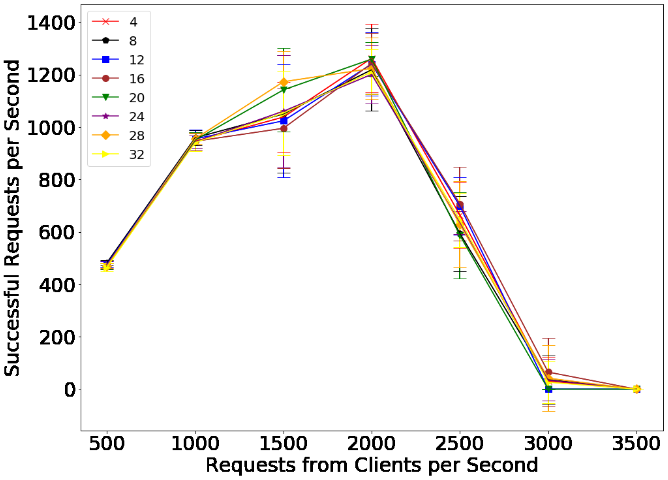

Fig. 2. **The throughput of a node with the increase of the request rate in Ripple.** The trend is similar to the HLF v0.6.

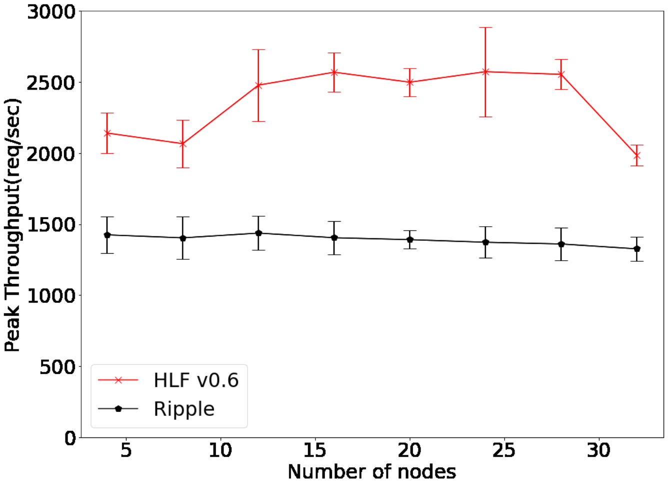

Fig. 3. **The peak throughput of a node with the increase of the number of nodes in HLF v0.6 and Ripple.**

However, after reaching the peak the throughput decreases sharply with the increase of the request rate, because the system cannot handle the demand.

Moreover, for either system, the trends of throughputs with different cluster sizes are similar, which means that the performance has no trend of increasing with the increase of nodes. In other words, consensus algorithms of Ripple, HLF v0.6 consensus, do not scale. Note that this limitation is well-known for classic BFT consensus algorithms because they solve a slightly more restrictive problem than the blockchain consensus [16]. In particular, both PBFT and BFT-SMaRt rely on a leader, which may act as a bottleneck [16].

C. Peak Throughput with the Increase of Nodes

The results of HLF v0.6 and Ripple consensus are depicted in Fig. 3, whereas Fig 4 depicts the results of HLF v1.0.

As expected, the peak throughput decreases with the increase of the cluster size in Ripple. The more nodes involved in a cluster, the more messages are exchanged. For a Ripple private network with only one validator, transactions are batched and sent to the validator, so that the validator should process numerous requests. Therefore, if the number of validators and machines running validators remains the same, the consensus does not scale.

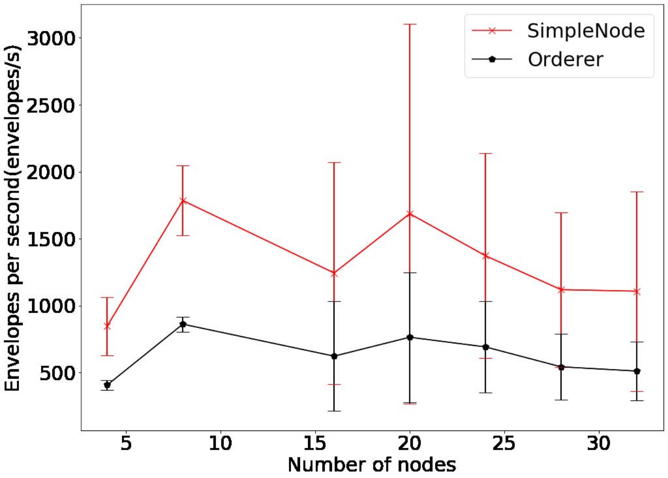

Fig. 4. **The peak throughput of a node with the increase of the number of nodes in HLF with BFT-SMaRt consensus.**

Meanwhile, the HLF v0.6 cluster's performance fluctuates with the increase of nodes, as well as the HLF with BFT-SMaRt consensus. Traditional PBFT consensus is used in HLF v0.6 so that the broadcast operation is always involved in the consensus, which creates multiple requests at a time in every node. The bottleneck effect at the leader prevents the consensus protocol from scaling.

As for BFT-SMaRt consensus in HLF, all traffic should go through the centralized nodes called orderers. Though the performance can scale by adding more orderers, it is impossible for the cluster to only have orderers but not have simple clients because of its nature. With the same number of orderers, the performance will not scale if the number of clients increases.

D. The Relation between Throughput and Latency

With the increase of the request rate, the latencies of both HLF v0.6 and Ripple first increase then decrease sharply, as shown in Fig 5 and Fig 6. The increase of the latency is because of the congestion control of the TCP protocol which prevents the computer from serving overwhelming requests simultaneously. The followed decrease is because systems are out of work. With TCP's underlying congestion control, if too much data comes to the server suddenly, the "slow start" mechanism will limit the reception rate on the server side and the "congestion avoidance" will half the congestion window if the network congestion is detected. Therefore, the high request rate will contribute to increasing latency because of triggering the network congestion.

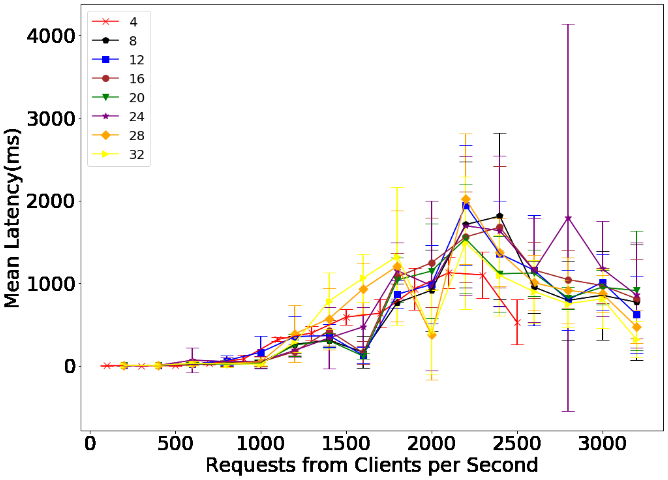

Fig. 5. **The throughput with the increase of the latency in HLF v0.6.**

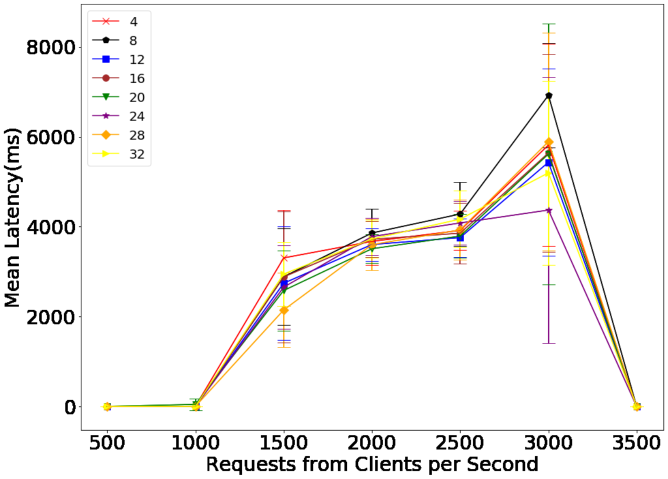

Fig. 6. **The throughput with the increase of the latency in Ripple.**

V. Limitations when Setting Up HLF v0.6

HLF v0.6 depends on Docker containers, where chaincodes are encapsulated and invoked by external requests. HLF clients communicate with chaincodes in Docker containers using Unix sockets or HTTP within the machine or endpoint. The experiment was launched on Ubuntu 16.04 OS. However, Ubuntu 16.04 adopts the `systemd` service manager rather than the previous `initd`, by which Docker starts as a service rather than applying configurations in `/etc/default/docker`. This causes the Unix socket endpoint to remain closed by default. Therefore, starting an HLF client with the same step as on previous versions of Ubuntu will fail on Ubuntu 16.04.

VI. Conclusion

This paper presents the first evaluation of Byzantine-tolerant blockchains that could be adapted for IoT. Our results are preliminary but show that more work is necessary for main secure blockchains to scale. This seems to confirm the well-known inadequacy of Byzantine consensus to large-scale blockchains, mainly because Byzantine consensus requires one proposal to be decided. A natural future work is thus the comparison of these blockchain solving the Byzantine consensus problem to recent scalable solutions that solve the Blockchain Consensus problem [16], like the Red Belly Blockchain.

Acknowledgment

We are grateful to Dinh Tien Tuan Anh, Mike Hibler, David Schwartz and Baohua Yang for their help. This research is in part supported under Australian Research Council's Discovery Projects funding scheme (project number 180104030) entitled "Taipan: A Blockchain with Democratic Consensus and Validated Contracts".

References

[1] S. Nakamoto, "Bitcoin: A peer-to-peer electronic cash system," 2008.

[2] V. Gramoli, "From blockchain consensus back to byzantine consensus," *Future Generation of Computer Systems*, 2017.

[3] C. Natoli and V. Gramoli, "The blockchain anomaly," in *Proceedings of the 15th IEEE International Symposium on Network Computing and Applications (NCA'16)*, 2016, pp. 310–317.

[4] ——, "The balance attack or why forkable blockchains are ill-suited for consortium," in *47th Annual IEEE/IFIP International Conference on Dependable Systems and Networks, DSN 2017, Denver, CO, USA, June 26-29, 2017*, 2017, pp. 579–590. [Online]. Available: https://doi.org/10.1109/DSN.2017.44

[5] M. Castro and B. Liskov, "Practical byzantine fault tolerance," in *Proceedings of the Third Symposium on Operating Systems Design and Implementation*, ser. OSDI '99. Berkeley, CA, USA: USENIX Association, 1999, pp. 173–186. [Online]. Available: http://dl.acm.org/citation.cfm?id=296806.296824

[6] A. Bessani, J. Sousa, and E. E. Alchieri, "State machine replication for the masses with bft-smart," in *Dependable Systems and Networks (DSN), 2014 44th Annual IEEE/IFIP International Conference on*. IEEE, 2014, pp. 355–362.

[7] D. Schwartz, N. Youngs, and A. Britto, "The ripple protocol consensus algorithm," *Ripple Labs Inc. White Paper*, vol. 5, 2014.

[8] T. T. A. Dinh, J. Wang, G. Chen, R. Liu, B. C. Ooi, and K.-L. Tan, "Blockbench: A framework for analyzing private blockchains," in *Proceedings of the 2017 ACM International Conference on Management of Data*, pp. 1085–1100.

[9] R. G. Brown, J. Carlyle, I. Grigg, and M. Hearn, "Corda: An introduction," *R3 CEV, August*, 2016.

[10] D. Ongaro and J. K. Ousterhout, "In search of an understandable consensus algorithm." in *USENIX Annual Technical Conference*, 2014, pp. 305–319.

[11] C. Cachin, "Architecture of the hyperledger blockchain fabric," in *Workshop on Distributed Cryptocurrencies and Consensus Ledgers*, 2016.

[12] J. Sousa, A. Bessani, and M. Vukolić, "A byzantine fault-tolerant ordering service for the hyperledger fabric blockchain platform," 2017.

[13] F. Armknecht, G. O. Karame, A. Mandal, F. Youssef, and E. Zenner, "Ripple: Overview and outlook," in *International Conference on Trust and Trustworthy Computing*. Springer, 2015, pp. 163–180.

[14] S. Popov, "The tangle," oct 2017.

[15] L. Baird, "The Swirlds Hashgraph Consensus Algorithm: Fair, Fast, Byzantine Fault Tolerance," pp. 1–27, 2016. [Online]. Available: http://www.swirlds.com/downloads/SWIRLDS-TR-2016-01.pdf

[16] T. Crain, V. Gramoli, M. Larrea, and M. Raynal, "Blockchain consensus," in *ALGOTEL 2017-19èmes Rencontres Francophones sur les Aspects Algorithmiques des Télécommunications*, 2017.

[17] S. Souders, "High-performance web sites," *Communications of the ACM*, vol. 51, no. 12, pp. 36–41, 2008.