PBFT vs Proof-of-Authority: Applying the CAP Theorem to Permissioned Blockchain *

Stefano De Angelis[1,2], Leonardo Aniello[1,2], Roberto Baldoni[1], Federico Lombardi[1,2], Andrea Margheri[2], and Vladimiro Sassone[2]

[1] Research Center of Cyber Intelligence and Information Security, Sapienza University of Rome
{deangelis;aniello;baldoni;lombardi}@dis.uniroma1.it
[2] University of Southampton
{stefano.de-angelis;f.lombardi;a.margheri;l.aniello;vsassone}@soton.ac.uk

Abstract

Permissioned blockchains are arising as a solution to federate companies prompting accountable interactions. A variety of consensus algorithms for such blockchains have been proposed, each of which has different benefits and drawbacks. *Proof-of-Authority* (PoA) is a new family of *Byzantine fault-tolerant* (BFT) consensus algorithms largely used in practice to ensure better performance than traditional *Practical Byzantine Fault Tolerance* (PBFT). However, the lack of adequate analysis of PoA hinders any cautious evaluation of their effectiveness in real-world permissioned blockchains deployed over the Internet, hence on an eventually synchronous network experimenting Byzantine nodes.

In this paper, we analyse two of the main PoA algorithms, named *Aura* and *Clique*, both in terms of provided guarantees and performances. First, we derive their functioning including how messages are exchanged, then we weight, by relying on the CAP theorem, *consistency*, *availability* and *partition tolerance* guarantees. We also report a qualitative latency analysis based on message rounds. The analysis advocates that PoA for permissioned blockchains, deployed over the Internet with Byzantine nodes, do not provide adequate consistency guarantees for scenarios where data integrity is essential. We claim that PBFT can fit better such scenarios, despite a limited loss in terms of performance.

1 Introduction

Blockchain is one of the most disruptive technologies of recent years. Firstly appeared as a decentralised public ledger for the Bitcoin cryptocurrency [19], blockchain is nowadays widely exploited to foster integration and federation among companies. Its distinguishing properties of data immutability, integrity and full decentralisation are key drivers for general purpose exploitations, ranging from Cloud computing to business-to-business applications.

Essentially, blockchain is a linked data structure replicated over a peer-to-peer network, where transactions are issued to form new blocks. Peers achieve distributed consensus on transaction ordering by placing them into new blocks; each block is linked to the previous via its hash. Such block creation process is carried out by distinguished nodes of the network, named *miners*, according to a distributed consensus algorithm. Besides cryptocurrency à la Bitcoin, miners support *smart contracts*, immutable programs deployed and executed in a decentralised fashion upon blockchain. Ethereum [26] was the first popularised smart contract framework.

Blockchain systems like Bitcoin and Ethereum are called *permissionless*, i.e. any node on the Internet can join and become a miner. Distributed consensus is here achieved via so-called *Proof of Work* (PoW), a computational intensive hashing-based mathematical challenge. PoW

*This work has been supported by the EU H2020 SUNFISH project, grant N.644666.

enjoys strong integrity guarantees and tolerates a sheer number of attacks [12], but this comes at a huge cost: lack of performance. This has led, together with the absence of privacy and security controls on data, to so-called *permissioned* blockchain, where an additional authentication and authorisation layer on miners is in place. Examples of permissioned blockchains are Multichain [18] and R3 Corda [9], (Hyperledger) Fabric [4] and permissioned-oriented Ethereum clients. Such systems have prompted federation of companies thus to facilitate their interactions without giving out guarantees on control and computation of data. By way of example, the Cloud Federation-as-a-Service solution [22] is exploiting a permissioned blockchain to underpin the governance of a federation of private Clouds [11] connected via the Internet.

Being the operating environment more trusted, permissioned blockchains rely on message-based consensus schema, rather than on hashing procedures. In such setting, dominant candidates are *Byzantine fault tolerant* (BFT) algorithms such as the Practical BFT (PBFT) [7]. Indeed, BFT-like algorithms have been widely investigated for permissioned blockchains [24] with the aim of outperforming PoW while ensuring adequate fault tolerance.

Proof-of-Authority (PoA) [20] is a new family of BFT algorithms which has recently drawn attention due to the offered performance and toleration to faults. It is currently used by Parity [23] and Geth [14], two well-recognised clients for permissioned setting of Ethereum. Intuitively, the algorithms operate in rounds during which an elected party acts as *mining leader* [15] and is in charge of proposing new blocks on which distributed consensus is achieved. Differently from PBFT, PoA requires less message exchanges hence provides better performance [10]. However, the actual consequences of such performance improvement is quite blurry, especially in terms of availability and consistency guarantees in a realistic *eventually synchronous* network model such as the Internet [5]).

To this aim, we take into account two of the main PoA implementations, named Aura [2] and Clique [8], which are used by Ethereum clients for permissioned-oriented deployments. In particular, the lack of appropriate documentation and analysis prevents from a cautious choice of PoA implementations with respect to provided guarantees, fault tolerance and network models.

In this paper, we first derive the actual functioning of the two PoA algorithms, both from the scarse documentation and directly from the source code, then we conduct a qualitative analysis in terms of the CAP theorem [13] (that is, in a distributed system only two out of *consistency*, *availability* and *partition tolerance* can be assured at the same time) and performance. The analysis assumes an eventually synchronous network and the presence of Byzantine nodes. The conducted analysis results that PoA algorithms favour availability over consistency, oppositely to what PBFT guarantees. In terms of latency, measured as the number of message rounds required to commit a block, PBFT lies in between Aura and Clique, outperforming the former and being worse than the latter. These results suggest that PoA algorithms are not actually suitable for permissioned blockchains deployed over the Internet, because they do not ensure consistency, and strong data integrity guarantees are usually the reason why blockchain-based solutions are employed. We advocate that PBFT is a better choice in this case, although its performance can be worse than some PoA implementations.

Outline. Section 2 introduces background concepts and comments the closest related work. Section 3 introduces the PoA consensus schema. Section 4 analyses PoA algorithms with respect to ensured guarantees and performance. Section 5 concludes and draws upon future works.

2 Background and Related Work

Consensus is a well-known problem of distributed computing. It consists in achieving an agreement among a distributed number of processes [5]. Among others, a prominent consensus

schema is so-called Byzantine Fault Tolerant (BFT). Protocols of such type are able to tolerate arbitrarily subverted nodes trying to hinder the achievement of an consistent agreement.

The *Practical Byzantine Fault Tolerance* (PBFT) [7] is one of the most well-established BFT algorithms. Specifically, it rests on three rounds of message exchange before reaching agreement. This ensures that $3f + 1$ nodes can achieve consensus also in presence of f Byzantine nodes; this is proved to be optimal [7]. Many other BFT algorithms have been proposed, mainly for improving PBFT performance; among others we can cite *Q/U*, *HQ*, *Zyzzyva*, *Aardvark*, see the survey in [24] for further details of each solution.

The wide interest on blockchain has prompted substantial research efforts on distributed consensus schema, specifically towards new ad-hoc BFT ones. In [16] the author reviews well-known families of consensus algorithms for both permissionless and permissioned blockchains. This includes Proof-of-Work (PoW), Prof-of-Stake (PoS), Delegated Proof-of-Stake (DPoS), Proof-of-Activity (PoW/PoS-hybrid), Proof-of-Burn (PoB), Proof-of-Validation (PoV), Proof-of-Capacity (PoC or Proof-of-Storage), Proof-of-Importance (PoI), Proof-of-Existence (PoE), Proof-of Elapsed Time (PoET), Ripple Consensus Protocol and Stellar Consensus Protocol (SCP). Although each algorithm is briefly described, they lack of any form of analysis in presence of Byzantine nodes under an eventual synchronous model.

Understanding the most appropriate consensus algorithm among the plethora before is a challenging task that a few works have tried to tackle. For instance, Sankar et al. investigates in [21] the main differences between SCP and consensus algorithms employed in R3 Corda and Hyperledger Fabric. Similarly, Mingxiao et al. extensively compare in [17] performances and security of PoW, PoS, DPoS, PBFT and Raft. While, Tuan et al. propose in [10] a practical benchmark for blockchain, named Blockbench, to systematically compare performances, scalability and security of multiple blockchain systems.

From a more formal perspective, Vukolić compares in [25] PoW with BFT-like approaches introducing the distinguishing property of *consensus finality*: the impossibility of reaching consensus without fully distributed agreement. In blockchain's jargon, it amounts to the impossibility of having forks. As expected, PoW does not enjoy consensus finality (as forks can happen), while all BFT-like approaches does (all parties reach agreement before consensus).

More related to permissioned blockchain, Cachin and Vukolić propose in [6] a thorough analysis of most-known permissioned systems and their underlying consensus algorithms in term of safety and liveness guarantees under eventual synchrony assumption. This work introduces for the first time a structured comparison among consensus algorithms, but it overlooks consistency and availability guarantees ensured by their usage; most of all it does not address PoA.

To sum up, none of the aforementioned works discuss the PoA consensus family. To the best of our knowledge, we believe this is the first work tackling the analysis of blockchain consensus algorithms from the perspective of the CAP theorem.

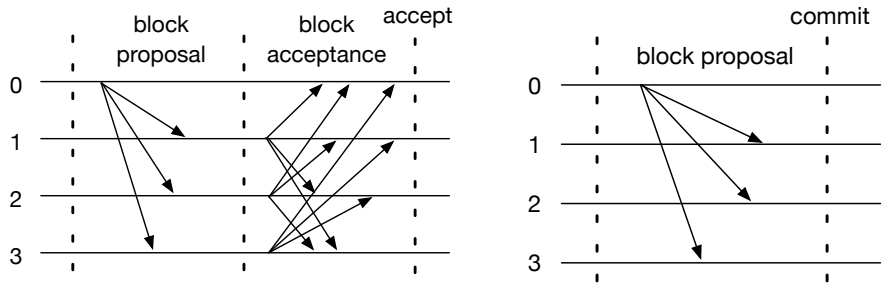
3 Proof-of-Authority Consensus

Proof of Authority (PoA) is a family of consensus algorithms for permissioned blockchain whose prominence is due to performance increases with respect to typical BFT algorithms; this results from lighter message exchanges. PoA was originally proposed as part of the Ethereum ecosystem for private networks and implemented into the clients *Aura* and *Clique*.

PoA algorithms rely on a set of N trusted nodes called the *authorities*. Each authority is identified by a unique *id* and a majority of them is assumed honest, namely at least $N/2 + 1$. The authorities run a consensus to order the transactions issued by *clients*. Consensus in PoA algorithms relies on a *mining rotation* schema, a widely used approach to fairly distribute the

responsibility of block creation among authorities [15, 11, 1]. Time is divided into *steps*, each of which has an authority elected as mining leader[1].

The two PoA implementations work quite differently: both have a first round where the new block is proposed by the current leader (*block proposal*); then Aura requires a further round (*block acceptance*), while Clique does not. Figure 1 depicts the message patterns of Aura and Clique, which will be detailed in next subsections.



(a) Aura (b) Clique

Figure 1: Message exchanges of Aura and Clique PoA for each step. In this example there are 4 authorities with *id* 0,1,2,3. The leader of the step is the authority 0.

3.1 Aura

Aura (Authority Round) [2] is the PoA algorithm implemented in Parity [23], the Rust-based Ethereum client. The network is assumed to be synchronous and all authorities to be synchronised within the same UNIX time t. The index s of each step is deterministically computed by each authority as $s = t/step\_duration$, where $step\_duration$ is a constant determining the duration of a step. The leader of a step s is the authority identified by the id $l = s \ mod \ N$.

Authorities maintain two queues locally, one for transactions Q_{txn} and one for pending blocks Q_b. Each issued transaction is collected by authorities in Q_{txn}. For each step, the leader l includes the transactions in Q_{txn} in a block b, and broadcasts it to the other authorities (*block proposal* round in Figure 1(a)). Then each authority sends the received block to the others (round *block acceptance*). If it turns out that all the authorities received the same block b, they *accept* b by enqueuing it in Q_b. Any received block sent by an authority not expected to be the current leader is rejected. The leader is always expected to send a block, if no transaction is available then an empty block has to be sent.

If authorities do not agree on the proposed block during the block acceptance, a voting is triggered to decide whether the current leader is malicious and then kick it out. An authority can vote the current leader malicious because (i) it has not proposed any block, (ii) it has proposed more blocks than expected, or (iii) it has proposed different blocks to different authorities. The voting mechanism is realised through a *smart contract*, and a majority of votes is required to actually remove the current leader l from the set of legitimate authorities. When this happens, all the blocks in Q_b proposed by l are discarded. Note that leader misbehaviours can be caused by benign faults (e.g., network asynchrony, software crash) or Byzantine faults (e.g., the leader has been subverted and behaves maliciously on purpose).

[1]For the cognitive, as there is no hash-based procedure like PoW, the consensus process is more appropriately called *minting*. However, for the sake of presentation, we will continue using the wording mining.

A block b remains in Q_b until a majority of authorities propose their blocks, then b is committed to the blockchain. With a majority of honest authorities, this mechanism should prevent any minority of (even consecutive steps) Byzantine leaders to commit a block they have proposed. Indeed any suspicious behaviour (e.g., a leader proposes different blocks to different authorities) triggers a voting where the honest majority can kick the current leader out, and the blocks they have proposed can be discarded before being committed.

3.2 Clique

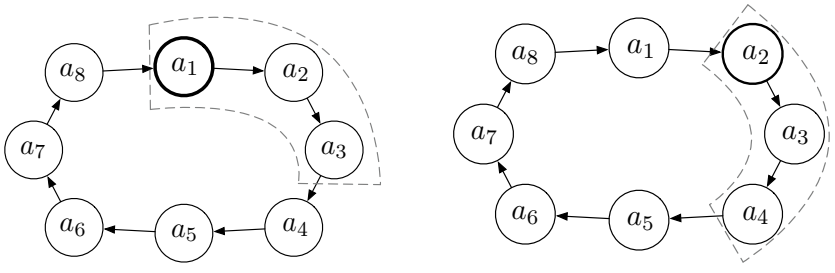
Clique [8] is the PoA algorithm implemented in Geth [14], the GoLang-based Ethereum client. The algorithm proceeds in *epochs* which are identified by a prefixed sequence of committed blocks. When a new epoch starts, a special *transition block* is broadcasted. It specifies the set of authorities (i.e., their ids) and can be used as snapshot of the current blockchain by new authorities needing to synchronise.

While Aura is based on UNIX time, Clique computes the current step and related leader using a formula that combines the block number and the number of authorities. Most of all, in addition to the current leader, other authorities are allowed to propose blocks in each step. To avoid that a single Byzantine authority could wreak havoc the network by imposing a sheer number of blocks, each authority is only allowed to propose a block every $N/2+1$ blocks. Thus, at any point in time there are at most $N-(N/2+1)$ authorities allowed to propose a block. Similarly to before, if authorities act maliciously (e.g., by proposing a block when they are not allowed) they can be voted out. Specifically, a vote against an authority can be casted at each step and if a majority is reached the authority is removed from the list of legitimate authorities.

As more authorities can propose a block during each step, forks can occur. However, fork likelihood is limited by the fact that each non-leader authority proposing a block delays its block by a random time, hence the leader block is likely to be the first received by all the authorities. If forks happen, the GHOST protocol [26] is used, which is based on a block scoring approach: leaders' blocks have higher scores, thus ensuring that forks will be eventually solved.

Figure 2 shows two consecutive steps and how current leader and authorities allowed to propose blocks change. There are $N=8$ authorities, hence $N-(N/2+1)=3$ authorities allowed to propose a block at each step, with one of them acting as leader (the bold node in Figure 2). In Figure 2(a), the a_1 is the leader while a_2 and a_3 are allowed to propose blocks. In Figure 2(b), a_1 is not allowed anymore to propose a block (it was in the previous step, so it has to wait $N/2+1$ steps), while a_4 is now authorised to propose and a_2 is the current leader.

Regarding the message exchange (see Figure 1(b)), at each step the leader broadcasts a



(a) Time t1 (b) Time t2

Figure 2: Selection of authorities allowed to propose blocks in Clique.

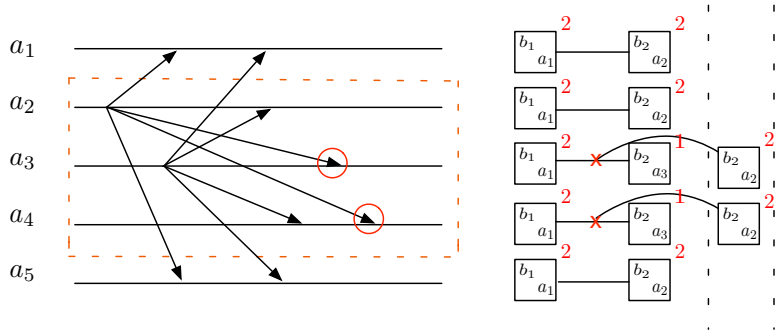


Figure 3: A fork occurring in Clique. Authority a_4 has the block proposed by a_3 as second block, while a_5 has the block proposed by a_2. Eventually, a_4 replaces the block proposed by a_3 with that proposed by a_2 because the latter has a higher score.

block and all the authorities directly commit it to the chain. By way of example, Figure 3 depicts a step where the leader authority a_2 proposes a new block, as well as the other allowed non-leader authority a_3. The former block precedes the latter in the views of a_1 and a_5, while the opposite occurs for a_4 and a_3. The resulting fork (see right-hand side of Figure 3) is easily detected by each authority when the next block is received, as it references a previous block not at disposal of the authority. By relying on the scoring mechanisms (i.e. blocks proposed by leaders win), the GHOST protocol resolves the forks.

4 Comparison of Aura, Clique and PBFT

Previous PoA algorithms are here compared with PBFT, first in terms of consistency and availability properties via the CAP Theorem (Section 4.1), then of performance (Section 4.2).

4.1 Consistency and Availability Analysis based on CAP Theorem

The CAP Theorem [3] states that in a distributed data store only two out of the three following properties can be ensured: *Consistency* (C), *Availability* (A) and *Partition Tolerance* (P). Thus any distributed data store can be characterised on the basis of the (at most) two properties it can guarantee, either CA, CP or AP. Before delving into the CAP-based analysis of the considered algorithms, we refine the definitions of these three properties in the context of permissioned blockchains deployed over the Internet, hence subjected to unforeseeable network delays of variable durations. In the following, we then assume an eventually synchronous network model where messages can be delayed among correct nodes, but eventually the network starts behaving synchronously and messages will be delivered (within a fixed but unknown time bound). This model is considered appropriate when designing real resilient distributed systems [6].

Consistency. A blockchain achieves consistency when forks are avoided. This property, as reported in Section 2, is referred to as consensus finality which, in the standard distributed system jargon, corresponds to achieving the *total order* and *agreement* properties of atomic broadcast. The latter is the communication primitive considered as the relevant type of consensus for blockchains [6]. When consistency cannot be obtained, we have to distinguish whether forks are resolved sooner or later (*eventual consistency*) or they are not (*no consistency*).

Availability. A blockchain is available if transactions submitted by clients are served and eventually committed, i.e. permanently added to the chain.

Partition Tolerance. When a network partition occurs, authorities are divided into disjoint groups in such a way that nodes in different groups cannot communicate each other.

Therefore, an Internet-deployed permissioned blockchain has to tolerate these adverse situations: (i) periods where the network behaves asynchronously; (ii) a (bounded) number of Byzantine authorities aiming at hampering availability and consistency. The maximum number of tolerated Byzantine nodes depends on the consensus algorithm: N/2 for PoA, N/3 for for PBFT. Since a blockchain must tolerate partitions, hence CA option is not considered, we analyse the algorithms with respect to CP and AP options. The results of this analysis are reported in Figure 4 and commented in the following.

Aura. Being based on UNIX synch time, authorities' clocks can drift and become out-of-synch. When authorities are distributed geographically over a wide area, resynchronization procedures cannot be effective due to network eventual synchrony. Hence, there can be periods where authorities do not agree on what is the current step and consequently on the current authority in force. Clocks' skews can be reasonably assumed strictly lower than the step duration, which is in the order of seconds, thus we can have short time windows where two distinct authorities are both considered as leaders by two disjoint sets of authorities, say $\mathcal{A}_1$ and $\mathcal{A}_2$. This can critically affect the consistency of the whole system.

Let $N_1 = |\mathcal{A}_1|$ and $N_2 = |\mathcal{A}_2|$ (where $N_1 + N_2 = N$ and N is an odd number) be the number of authorities in the two sets, respectively. We have a majority of authorities, say $\mathcal{A}_1$, agreeing on who is the current leader. This leads to a situation as depicted in Figure 5: authorities in $\mathcal{A}_1 = \{a_1, a_3, a_5\}$ see steps slightly out of phase with respect to the authorities in $\mathcal{A}_2 = \{a_2, a_4\}$. Indeed, the time windows coloured in grey are those where $\mathcal{A}_1$ disagrees with $\mathcal{A}_2$ on who is the current leader. During time window W_1, a_2 considers itself the leader and sends a block to the other authorities. a_2 is believed to be the leader by the authorities in $\mathcal{A}_2$ but not by those in $\mathcal{A}_1$, hence the former authorities accept its block while the latter ones reject it. During the time window W_2, authorities in $\mathcal{A}_1$ expect a_2 to send a block but this does not occur because it has already sent its block for the current step: at the end of W_2 authorities in $\mathcal{A}_1$ will vote a_2 as malicious and being a majority they will force a_2 to be removed. Therefore, all the remaining authorities in $\mathcal{A}_2$ will be eventually voted out one by one analogously. As $\mathcal{A}_2$ is a minority, its authorities are voted out in a number of steps lower than that required to commit enqueued

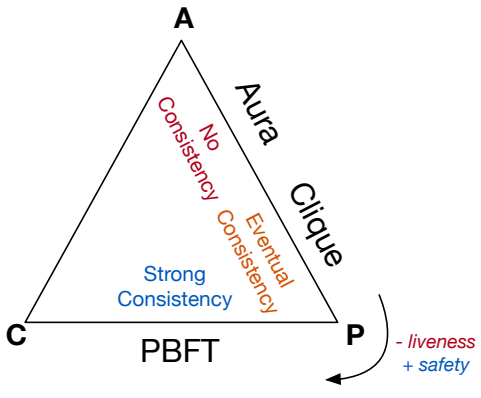


Figure 4: Classification of Aura, Clique and PBFT according to the CAP Theorem.

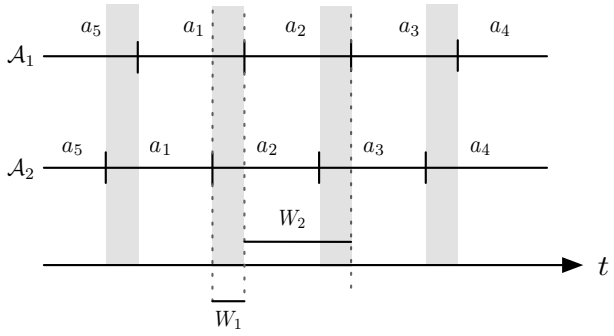


Figure 5: Example of out-of-synch authorities in Aura (each step for a set of authorities is labelled by the expected leader).

blocks (see Section 3.1), hence there will be no different views of the chain and the consistency is preserved. It is to note that in this case all the authorities are honest.

The same situation can affect consistency when authorities are Byzantine. Let us consider a scenario where there are B malicious authorities, all in $\mathcal{A}_1$ and, differently from before, they do not vote against authorities in $\mathcal{A}_2$. If $B \geq N_1 - N/2$, then a majority is not reached to vote out authorities in $\mathcal{A}_2$, hence the blocks they have proposed achieve finality and are committed to their local chains. This causes a fork that is never resolved: if authorities are not voted out, their blocks are consider as valid and part of the chain. Therefore, a minority of Byzantine authorities is sufficient to realise this attack (see Appendix A) and causes *no consistency* in the system. Anyway, transactions keep being committed over time regardless of what a minority of Byzantine authorities do. Hence, Aura can be classified as an AP system, with no consistency guarantees.

Clique. By design, Clique allows more than one authority to propose blocks with random delays. This permits coping with leaders that could not have sent any block due to either network asynchrony or benign/Byzantine faults. Resulting forks are anyway resolved by the Ethereum GHOST protocol, hence we have *eventual consistency*. On the other hand, as the mining frequency of authorities is bounded by $\frac{1}{N/2+1}$, a majority of Byzantine authorities is required to take over the blockchain. This PoA algorithm can thus be classified as AP, with eventual consistency guarantees.

PBFT. As long as less than one third of nodes are Byzantine, PBFT has been proved to guarantee consistency, i.e. no fork can occur [7]. Because of the eventual synchrony of the network, the algorithm can stall and blocks cannot reach finality. In this case, consistency is preserved while availability is given up. PBFT can then be easily classified as a CP system according to the CAP theorem.

4.2 Performance Analysis

The analysis here reported is qualitative and only based on how the consensus algorithms work in terms of message exchanging. The performance metrics usually considered for consensus algorithms are transaction latency and throughput. In the specific case of permissioned blockchains, we measure the latency of a transaction t as the time between the submission of t by a client and the commit of the block including t. Contrary to CPU intensive consensus algorithms such as PoW, here we can safely assume that latency is communication-bound rather than CPU-

bound, as there is no relevant computation involved. Hence, we can compare the algorithms in terms of the number of message rounds required before a block is committed. Evaluating the throughput at a qualitative level is much more challenging, as it closely depends on the specific parallelisation strategy (e.g., pipelining) employed by each algorithm implementation. Thus, we deem more correct to compare throughput performance by the means of proper experimental evaluations that we plan to carry out as future work.

We assess how many message rounds are required for each algorithm in the normal case, i.e. when no condition occurs that makes any corner case to be executed. For example, for Aura we do not consider the situation when some authorities suspect the presence of subverted nodes and trigger a voting.

In Aura, each block proposal requires two message rounds: in the first round the leader sends the proposed block to all the other authorities, in the second round each authority sends the received block to all the other authorities. A block is committed after a majority of authorities have proposed their blocks, hence the latency in terms of message rounds in Aura is $2(N/2+1)$, where N is the number of authorities.

In Clique, a block proposal consists of a single round, where the leader sends the new block to all the other authorities. The block is committed straight away, hence the latency in terms of message rounds in Clique is 1. Such a huge difference between Aura and Clique is due to their different strategies to cope with malicious authorities aiming at creating forks: Aura waits that enough other blocks have been proposed before committing, Clique commits immediately and copes with possible forks after they occur. Clique seems to outperform PBFT too, which takes three message rounds to commit a block.

5 Conclusion

In this paper we derive the functioning of two prominent consensus algorithms for permissioned blockchains based on the PoA paradigm, namely Aura and Clique. We provide a qualitative comparison of them with respect to PBFT in terms of consistency, availability and performance, by considering a deployment over the Internet where the network is realistically modelled as eventually synchronous rather than synchronous.

By applying the CAP Theorem, we claim that in this setting PoA algorithms can give up consistency for availability when considering the presence of Byzantine nodes. This can prove to be unacceptable in scenarios where the integrity of the list of transactions has to be absolutely kept (which is likely to be the actual reason why a blockchain-based solution is used). On the other hand, PBFT keeps the blockchain consistent at the cost of availability, even when the network behaves temporarily asynchronously and Byzantine nodes are present; this behaviours is much more desirable when data integrity is a priority. Despite one of the most praised advantages of PoA algorithms is their performance, our qualitative analysis shows that in terms of latency the expected loss of PBFT is bounded, and can be offset by the gain in consistency guarantees.

As future work, we plan to deepen the analysis of PoA algorithms by engaging further reverse engineering tasks and thorough experimental evaluations. The final goal is to validate and possible revise our claims on the availability and consistency guarantees of PoA and PBFT, by implementing the adverse scenarios we envisioned in Section 4.1. Furthermore, we want to collect real performance measurements, both transaction latency and throughput, and to test scalability with respect to varying input transaction rates and number of nodes/authorities. Moreover, we are moving towards a formalisation of permissioned blockchains so to define a framework for benchmarking and evaluating these algorithms with a more formal approach.

References

[1] L. Aniello, R. Baldoni, E. Gaetani, F. Lombardi, A. Margheri, and V. Sassone. A prototype evaluation of a tamper-resistant high performance blockchain-based transaction log for a distributed database. In *EDCC*. IEEE, 2017.

[2] Aura. https://github.com/paritytech/parity/wiki/Aura.

[3] E. Brewer. Cap twelve years later: How the "rules" have changed. *Computer*, 45(2):23–29, 2012.

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

[5] C. Cachin, R. Guerraoui, and L. Rodrigues. *Introduction to reliable and secure distributed programming*. Springer, 2011.

[6] C. Cachin and M. Vukolić. Blockchains consensus protocols in the wild. *arXiv preprint arXiv:1707.01873*, 2017.

[7] M. Castro and B. Liskov. Practical byzantine fault tolerance and proactive recovery. *ACM Trans. Comput. Syst.*, 20(4):398–461, 2002.

[8] Clique. https://github.com/ethereum/EIPs/issues/225.

[9] R3 Corda. https://github.com/corda/corda/.

[10] 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 *SIGMOD*, pages 1085–1100. ACM, 2017.

[11] E. Gaetani, L. Aniello, R. Baldoni, F. Lombardi, A. Margheri, and V. Sassone. Blockchain-based database to ensure data integrity in cloud computing environments. In *ITA-SEC*, volume 1816. CEUR-WS.org, 2017.

[12] J. A. Garay, A. Kiayias, and N. Leonardos. *The Bitcoin Backbone Protocol: Analysis and Applications*, volume 9057 of *LNCS*, pages 281–310. Springer, 2015.

[13] S Gilbert and N. Lynch. Brewer's conjecture and the feasibility of consistent, available, partition-tolerant web services. *Acm Sigact News*, 33(2):51–59, 2002.

[14] Go-Ethereum. https://geth.ethereum.org.

[15] BitFury Group and J. Garzik. Public versus private blockchains. *White Paper*, 2015. http://bitfury.com/content/5-white-papers-research/public-vs-private-pt1-1.pdf.

[16] J. Mattila. The blockchain phenomenon. *Berkeley Roundtable of the International Economy*, 2016.

[17] D. Mingxiao, M. Xiaofeng, Z. Zhe, W. Xiangwei, and C. Qijun. A review on consensus algorithm of blockchain.

[18] Multichain. https://www.multichain.com/.

[19] S. Nakamoto. Bitcoin: A peer-to-peer electronic cash system, 2008. Available at https://bitcoin.org/bitcoin.pdf.

[20] Proof of Authority. https://github.com/paritytech/parity/wiki/Proof-of-Authority-Chains.

[21] L. S. Sankar, M. Sindhu, and M. Sethumadhavan. Survey of consensus protocols on blockchain applications. In *ICACCS*, pages 1–5. IEEE, 2017.

[22] F. P. Schiavo, V. Sassone, L. Nicoletti, and A. Margheri. FaaS: Federation-as-a-Service, 2016. Technical Report. Available at https://arxiv.org/abs/1612.03937.

[23] Parity Technologies. https://www.parity.io.

[24] L. Tseng. Recent results on fault-tolerant consensus in message-passing networks. In *Int. Colloquium on Structural Information and Communication Complexity*, pages 92–108. Springer, 2016.

[25] M. Vukolić. The quest for scalable blockchain fabric: Proof-of-work vs. bft replication. In *International Workshop on Open Problems in Network Security*, pages 112–125. Springer, 2015.

[26] G. Wood. Ethereum: A secure decentralised generalised transaction ledger. *Ethereum Project Yellow Paper*, 2014.

A Proof of Sufficiency of a Minority in the Majority to prevent the Majority

Let us consider a set $\mathcal{S}$ with an odd number of elements $N = 2K + 1$, and a partition of such set in two non-empty subsets $\mathcal{S}_1$ and $\mathcal{S}_2$ with cardinality N_1 and N_2, respectively, such that $\mathcal{S}_1$ is a majority and $\mathcal{S}_2$ a minority, i.e.,

$$K + 1 \leq N_1 \leq 2K \tag{1}$$

$$1 \leq N_2 \leq K$$

$$N_1 + N_2 = N$$

We want to prove that it suffices to remove a minority [2] of B elements from $\mathcal{S}_1$ to make it become a minority. Hence, we want to prove that

$$\exists B \mid N_1 - B \leq K \wedge B \leq K \tag{2}$$

Proof. Equation 2 can be proved by demonstrating that $N_1 - K \leq K$. This expression can be written as $N_1 \leq 2K$, which is always verified because of Equation 1.

□

[2] A minority with respect to the set $\mathcal{S}$.