Towards An Analysis of Network Partitioning Prevention for Distributed Ledgers and Blockchains (Author's Pre-Publication Version)

Michael Kuperberg
Blockchain and Distributed Ledgers Group
DB Systel GmbH
Frankfurt am Main, Germany
michael.kuperberg@deutschebahn.com

***Abstract*—Distributed Ledgers and Blockchains involve decentralized networking technologies such as peer-to-peer networks. Both permissioned and unpermissioned blockchains are designed to tolerate and to overcome the failure of individual nodes, which need to update themselves once they reconnect to the blockchain network.**

At the same time, it is important to avoid accidental network partitioning, since partitioning can lead to inconsistent "multiple truths" and violate the shared consensus about the global state, which must be maintained across all active and re-emerging nodes. In general, reconciling inconsistent "multiple truths" would require the deletion of information which was persisted to the blockchain (in one of the network partitions), and such removal violates the core blockchain principle of audit-proof and tamper-proof "write once read many" access (WORM).

However, current mainstream implementations such as Ethereum do not protect against network partitioning (either by accident or caused by an attack), and there is no research on how the consensus implementations behave in the presence of network partitioning. Given the differences in design and implementation across ledgers and blockchains, a systematic analytic approach must be established before partitioning prevention in specific products is studied.

The contribution of this paper is a foundational analysis of enterprise-grade consensus protocols, including design recommendations for partitioning avoidance for Proof-of-Authority in Hyperledger Fabric. We also survey related work and lay out next research steps.

***Index Terms*—blockchain, distributed ledger, network partitioning, network bisection, decentralized consensus**

I. Introduction and Problem Statement

Blockchains are distributed and decentralized ledgers, storing the data across many networked nodes without a central all-powerful authority (see [1] for a detailed introduction). Many blockchain products exhibit a WORM (write once, read many) behaviour, usually through an append-only implementation which concatenates data blocks using hashes. "Smart Contracts" are blockchain-stored executable rules; the execution results are also stored on the blockchain.

Blockchains are designed for resiliency and failure tolerance, so that individual failed nodes or offline nodes do not stop the network. The details of the resilience policies vary widely across implementations: for example, permissioned blockchains might prohibit anonymous nodes and control the number of nodes per authenticated participant. Restrictions also apply for the networking aspects - even though decentralization means that peer-to-peer architectures are prevalent.

However, decentralization and failure tolerance also mean that naive implementations of peer-to-peer blockchains can be subject to network partitioning: if two (or more) groups of nodes become fully isolated from each other, each isolated group can emerge and see itself as a complete, autonomous and fully-functioning network. Each group then continues to establish its own consensus about data state - without being aware of the other groups. From a global, group-overarching perspective, this leads to "multiple truths", with a high chance of conflicts and contradictions.

The CAP theorem [2] states that it is not possible to achieve more than two out of three following qualities at the same time: Consistency, Availability and Tolerance of Network Partitioning. The applicability of CAP to Distributed Ledger Technologies (DLTs) is being discussed (see e.g. [3]–[6]), and conflicting opinions exist. In practice, at least one of the properties is (partially) sacrificed. For example, forks (branches) creating during network partitioning phases can be either tolerated and kept (even with conflicting data), or the data can be reconciled/discarded after the network merges again. However, accepting conflicting truths on a single blockchain runs counter to its design intent.

The contribution of this paper is a foundational analysis of the network partitioning problem for blockchains, with an emphasis on enterprise-grade consensus protocols such as Proof-of-Authority, Proof-of-Stake and Proof-of-Work. We also include design recommendations for partitioning *avoidance* in Proof-of-Authority in Ethereum and Hyperledger Fabric.

The paper is structured as follows: we study related work (Sec. II), describe how selected established consensus algorithms can contribute to partitioning prevention (Sec. III), and lay out further investigation steps for future work (Sec. IV).

II. Related Work

Network partitioning for distributed databases has been addressed decades ago, e.g. in [7]. Considered as a Byzantine fault, partitioning was subject to analysis of consensus algorithms long before blockchain: for example, in 1982, Fischer et al. [8] showed for the "all must consense" situation that involves an *asynchronous* system of processes (some of which may be unreliable) that "every protocol for this problem has the possibility of nontermination, even with only one faulty process". The authors note that solutions are known for the synchronous case.

In [9], Ekparinya et al. describe an attack (based on identity/node "clones") against Proof-of-Authority (PoA) for Ethereum. They also propose countermeasures to ensure that the network can remain live and safe. However, their results are only discussed in the context of Ethereum PoA implementations, whereas other blockchain products (e.g. Hyperledger Fabric) have substantially different consensus implementations.

In [10], Singhal and Masih discuss different methods for scaling blockchain technology for automotive uses. While the authors mention the effects of network partitioning onto different blockchains (especially those based on DAGs, such as IOTA) and onto consensus algorithms (especially PFBT), they do not discuss any protection measures.

In [11], Ekparinya et al. demonstrate attacks on Ethereum using network partitioning, but do not discuss how to protect against them. In [12], Dalui et al. deliberately *introduce* network partitioning to reduce the message exchange overhead - however, [12] does not offer a solution for dealing with *undesirable* network partitioning. In [13], Buntinas describes consensus for highly parallel applications using MPI (Message Passing Interface) architectures, but does not consider network partitioning. Network partitioning for blockchains has been studied by Saito and Yamada in [14], but they do not discuss Proof-of-Authority and do not provide suggestions on choosing a partitioning-resistant consensus. In [15], Anjum et al. list IPFS as "highly resilient against network partitioning", but do not consider this characteristic for the other analyzed technologies (which include Ethereum, Hashgraph etc.).

In [16], Wang et al. highlight the role of network partitioning for IoT-oriented blockchains and discuss the connection-monopolizing Eclipse attacks, but do not discuss how consensus protocols or specific blockchain implementations are affected. In [17], Hu et al. discuss the threat of network partitioning as part of the security analysis for a delay-tolerant payment scheme based on the Ethereum blockchain. The authors propose a solution which employs a centralized proxy to "deploy a smart contract that tracks node connections and detects network partitioning". Additionally, they propose that the blockchain nodes monitor their P2P connections and "freeze" transaction processing "until the overlay network stabilizes again". However, these proposals have not been implemented and the authors do not discuss the consensus-dependent effects of network partitioning.

In [18], Chan et al. propose a blockchain design that is built around "pipelined BFT". The authors claim that the resulting solution is "arbitrarily partition tolerant and resists any adversary that controls less than 1/3 of corrupt nodes". However, [18] does not analyze PoW, PoS or PoA, and does not provide insights into mainstream blockchain products.

In [19], Homoliak et al. introduce a security reference architecture for blockchains, and include network partitioning into the list of threats. While the authors show which further attacks are enabled by network partitioning, they do not present any strategies to protect against this threat.

In [20], Keshav et al. present a resilient consensus algorithm called RCanopus that "guarantees safety even in the presence of Byzantine attacks and network partitioning". However, RCanopus requires a specific, hierarchical node topology, and as of June 2019, it remains to be implemented - thus, its applicability to mainstream blockchain implementations such as Hyperledger Fabric or Ethereum is currently unclear.

In [21], Li et al. deal with securing PoS blockchain protocols. In [22], Yu et al. propose a novel consensus protocol. In both papers, the authors explain the risks of network partitioning, but do not provide a solution.

Partitioning prevention on the network level (independently from consensus) has also been studied, e.g. in [23] and in [24].

III. Partitioning Prevention in Mainstream Consensus Algorithms

In blockchains, the agreement about the current state (and the updates that alter it) is decided in a collaborative manner, rather than in a top-down and/or centralized way. The agreement is achieved through consensus algorithms, which are designed to withstand a defined maximum of ill-behaving or non-responding participants. Different consensus algorithms exist in practice, for comparisons see e.g. [25], [26] and others.

A blockchain application consists of several layers, which can be grouped as shown in Fig. 1.

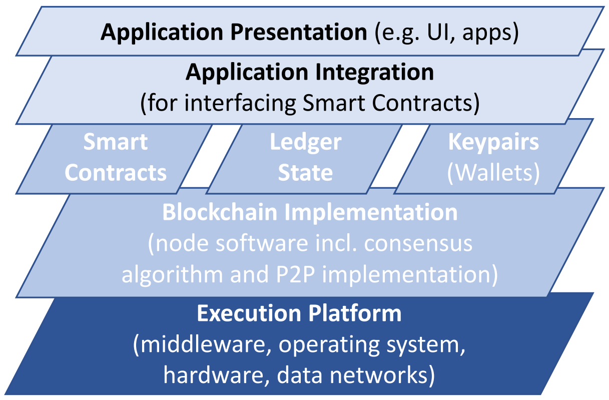
Application Presentation (e.g. UI, apps)
Application Integration
(for interfacing Smart Contracts)
Smart Contracts
Ledger State
Keypairs (Wallets)
Blockchain Implementation
(node software incl. consensus algorithm and P2P implementation)
Execution Platform
(middleware, operating system, hardware, data networks)

Fig. 1. Layers of a blockchain application incl. the runtime platform

The blockchain community often considers "Proof of Work" (PoW), "Proof of Authority" (PoA), "Proof of Stake" (PoS) as leading mainstream enterprise-grade consensus algorithms. There exist many more algorithms such as RAFT, Paxos, Tendermint, "Proof of Elapsed Time" (PoET), "Proof of Space" (PoSpace), "Pure Proof of Stake" (PPOS) and others, but these are mainly either niche or research-grade or being productized.

We thus limit our attention to PoA, PoS and PoW; note that these three solutions patterns (we refer to them as Po*) only perform certain parts of the consensus: Po* exclude the preceding transaction proposal steps and do not cover the following step: the solution-specific propagation of agreed changes to data.

Instead, Po* define which network participants *can* create a new block, and also define which network participants *will* create a new block that the network will (try to) agree upon. The subtleties of the individual Po* implementations are outside the scope of this paper; we focus on their architectural capability in the context of network partitioning.

A. PoA in Hyperledger Fabric 1.4

Hyperledger Fabric is an example of an enterprise-grade permissioned DLT-based blockchain which exclusively employs PoA instead of other consensus algorithms. In Hyperledger Fabric,

- a *channel* groups a subset of the participating organizations into a logical "conversation context" and limits the distribution of the data from the channel's transactions
- an *endorsement policy* defines which organisation must inspect and confirm a transaction proposal before it can be persisted to the ledger (the policy can be customized both for channels and for the network)
- a *client node* submits a *transaction proposal* (on behalf of its parent *organization*)
- the *endorser peer nodes* of the channel-participating organizations inspect the transaction proposal by locally simulating the transaction proposal; their signed findings (validity checks) are reported to the *ordering service* via the client nodes, following the endorsement policy
- within the ordering service (which has a replicated and resilient multi-node setup), the *leader* orderer node checks the client's privileges and the endorsers' findings (according to the predefined endorsement policy); if the policy and the endorsements match, the ordering service adds the transaction to a forthcoming block
- the ledger-holding peer nodes update their replicates with the new blocks, after having checked them for consistency (e.g. for being signed by the orderer and by the endorsing peers)

In other words, the orderer does not perform complete consistency checks; even after the orderer has put a transaction into a block, the data-replicating peers of the organization double-check the block before adding it to their copy.

Beyond the specific implementation in Hyperledger Fabric, the generalized PoA design can separate the roles in a slightly different way, both at the *block granularity* and at *transaction granularity*. Specifically, among the nodes in the subset, two separate selection policies exist to dynamically select all *or* some of the authorities to perform the following PoA actions for each transaction/block:

1) propose a new transaction to the network
2) inspect/verify and confirm ("endorse") a new transaction/block
3) serialize endorsed transaction into a block that is appended to the ledger

For 2), even if the network is not partitioned after 1) has ended, it depends on the inspection/confirmation policy whether network partitioning will be prevented: in general (independent of the endorsement policy), if and only if the policy of "all authorities must confirm" is in place, a network partitioning will be detected in 2) on the consensus/protocol level (the partitioning might be detected earlier on network level). This (very general) statement is valid for a setup with 2 authorities or even 1, but it can be relaxed for setups with 3 or more authorities, as this all-must confirmation policy bears its own risks: if a single authority goes offline, the network operations are suspended. Therefore, for practicability purposes, a (custom-defined) "majority" subset of authorities is often more likely to be the endorsing subset - which is still better than the probabilistic majority handling in unpermissioned PoW networks, and also better in terms of energy efficiency.

To analyze the endorsement policy "at least m out of a authorities must endorse" in a network with n organizations (i.e. $0 < m < a \leq n$), we start a bisection case (partitioning with $p = 2$, i.e. the network consists of p_1 and p_2). Note that it means $n \geq 2$ and that despite $n \geq a$, non-authorities cannot play an active role during PoA consensus: thus, *at consensus level*, they cannot actively contribute to protecting the blockchain network from the effects of network partitioning.

To start with a simpler discussion, we also make an initial assumption that each authority owns only one endorsing node and leave multiple endorsing nodes per authority to future work. We do not place restrictions of the number of ordering nodes within the ordering service, since a single active orderer is sufficient to perform the ordering step (which comes after 2)). We also assume that no additional orderers or endorsing peers are set up or torn down *dynamically* during bisection. Note that despite these simplifications and even if p_1 contains m endorsing nodes, p_2 can still contain $a-m$ endorsing nodes and if $(a-m) \geq m$, then the endorsement can be reached within partition p_2 even when it is disconnected from p_1.

Based on these assumptions, certain scenarios and behaviours worth analyzing with can be derived based on a cross product of the following possibilities:

- 4 possibilities for e_{p_1} (Endorser nodes in p_1):
  $e_{p_1} = 0$ or
  $0 < e_{p_1} < m$ or
  $m \leq e_{p_1} < a$ or
  $e_{p_1} = a$
- Likewise, 4 possibilities for e_{p_2} (Endorser nodes in p_2): $e_{p_2} = 0$ or $0 < e_{p_2} < m$ or $m \leq e_{p_2} < a$ or $e_{p_2} = a$
- 2 possibilities for ord_{p_1} (orderer nodes in p_1): $ord_{p_1} = 0$ or $ord_{p_1} > 0$
- Likewise 2 possibilities for ord_{p_2} (orderer nodes in p_2): $ord_{p_2} = 0$ or $ord_{p_2} > 0$

Obviously, not all $4 \times 4 \times 2 \times 2$=64 combinations can be set up: for example, it is impossible to combine $e_{p_1} = a$ with $e_{p_2} = a$, and it is impossible to combine $ord_{p_1} = 0$ with $ord_{p_2} = 0$ (since that the second combination makes the network ordererless). Thus, there are 3 possible combinations for ord_{p_1} with ord_{p_2}. Likewise, there are 4 possible combinations of e_{p_1} with e_{p_2}, after symmetric cases are removed:

1) $e_{p_1} = 0$ (none) with $e_{p_2} = a$ (all)
2) $0 < e_{p_1} < m$ with $0 < e_{p_2} < m$
3) $m \leq e_{p_1} < a$ with $0 < e_{p_2} < m$
4) $m \leq e_{p_1} < a$ with $m \leq e_{p_2} < a$

Case	Endorser nodes in p_1	Endorser nodes in p_2	Orderer nodes in p_1	Orderer nodes in p_2
A	$e_{p_1} = 0$ (none)	$e_{p_2} = a$ (all)	0	> 0
B	$e_{p_1} = 0$ (none)	$e_{p_2} = a$ (all)	> 0	0
C	$e_{p_1} = 0$ (none)	$e_{p_2} = a$ (all)	> 0	> 0
D	$0 < e_{p_1} < m$	$0 < e_{p_2} < m$	0	> 0
E	$0 < e_{p_1} < m$	$0 < e_{p_2} < m$	> 0	0
F	$0 < e_{p_1} < m$	$0 < e_{p_2} < m$	> 0	> 0
G	$m \leq e_{p_1} < a$	$0 < e_{p_2} < m$	0	> 0
H	$m \leq e_{p_1} < a$	$0 < e_{p_2} < m$	> 0	0
J	$m \leq e_{p_1} < a$	$0 < e_{p_2} < m$	> 0	> 0
K	$m \leq e_{p_1} < a$	$m \leq e_{p_2} < a$	0	> 0
L	$m \leq e_{p_1} < a$	$m \leq e_{p_2} < a$	> 0	0
M	$m \leq e_{p_1} < a$	$m \leq e_{p_2} < a$	> 0	> 0

TABLE I
12 CONSTELLATIONS FOR PoA NETWORK PARTITIONING WITH a AUTHORITIES, "AT LEAST m OUT OF a" ENDORSING POLICY WITH $m < a$, EXACTLY ONE ENDORSING NODE FOR EACH AUTHORITY AND TWO PARTITIONS ($p = 2$) SO THAT $e_{p_2} = (a - e_{p_1})$

In Table I, we provide a condensed listing of the $4 \times 3 = 12$ possible constellations, removing "symmetric" cases such as those with "$e_{p_1} = a$ and $e_{p_2} = 0$" (which is a permutation of cases A through C), since the "symmetric" cases lead to the same conclusions as their siblings listed in Table I.

- Case A: consensus can be found in p_2 but not in p_1; if this situation persists then p_1 (which can contain non-endorsing peers with ledger replicas) will be left with stale data but will update itself once the network reunites. A node in p_1 can detect network partitioning when non-endorsing peers fail to detect their organisations' endorsing peers, if an organisation's node topology is monitored and jointly known, or if no new blocks are created for an unusually long period of time even though transaction proposals are made. A node in p_2 (incl. endorsing peers) can only detect network partitioning when they were already aware of those non-endorsing peers that ended up in p_1 - however, consensus in p_2 can run being unaware of the network partitioning.
- Case B: consensus cannot be found, neither in p_1 nor in p_2 and thus no stale data can emerge; the network is not able to serve write requests until it reunites but is able to serve read requests. Network partitioning can be detected in p_2 (since consensus attempts fail after endorsement but before ordering). Network partitioning can be detected in p_1 if the ordering service finds that it has no endorsers at all despite having endorsement policies (which always require > 1 endorsements).
- Case C: this is identical to Case A with the difference that the orderer node in p_1 can detect network partitioning when it finds that p_1 has no endorsing peers at all.
- Cases D, E and F: consensus cannot be found, neither in p_1 nor in p_2 and thus no stale data can emerge. Detection of network partitioning is possible from within p_1 (cases E and F) and p_2 (cases D and F) using the observations of orderers.
- Cases G, H and J: consensus cannot be established in p_2 while for H and J, consensus can be established in p_1 (for G, there is no orderer in p_1 to create consensus there). In all three cases, no stale data can emerge in p_1 because p_2 cannot create data at all. Detection of network partitioning might be possible from within p_1 (cases H and J) and p_2 (cases G and J) using the observations of orderers. Once the network reunites, ledger replicas in p_2 (which can have stale data in cases H and J) can be updated to match consensus-found data in p_1, if any.
- Case K: consensus can be found in p_2 but cannot be established in p_1 and thus no conflicting data can be created in p_1 (but stale data can emerge in p_1). This case can be reasoned about as above w.r.t. reunification and re-sync.
- Case L: consensus can be found in p_1 but cannot be established in p_2 and thus no conflicting data can be created in p_2 (but stale data can emerge in p_2). This case is mirror-inverted to K.
- Case M: consensus can be found both in p_1 and in p_2 because both partitions contain enough endorsers ($\geq m$) and also > 0 orderers, i.e. both partitions are self-sufficient. This is the most undesirable case as parallel truths can be established in the partitions with the risk that conflicting data is established which cannot be reconciled; stale data can emerge as well. However, partitioning can be detected in both p_1 and p_2 if the entire set of a authorities is known to the orderers, and the processing of write requests could be halted in each partition - yet this would halt the processing of write requests in the entire blockchain network.

Still, the bisection scenario in case M can be avoided by prefering consistency over liveliness and imposing the constraint

$$2 \times m > a$$

upon the endorsement policy (for the bisection case with $p = 2$). This formula also holds for a general partitioning scenario with $p \geq 2$, since the above constraint prohibits > 1 concurrent consensus findings no matter how many partitions exist.

It should also be noted that aforementioned PoA implementation in Hyperledger Fabric puts full trust into an orderer node, which must not accept a transaction proposal which lacks endorsements. Clearly, operating a single endorser node (which is technically possible) incurs reliability risks and trust issues. However, while multiple orders are supported (using the Apache Kafka middleware or RAFT), the actual ordering decision is still made by one orderer node only, the so-called "leader". Fabric design is anchored at the "organization" perspective, and it is expected that each organization takes its own steps for reliability and failover; it is not expected that a Fabric organization's infrastructure fails completely so all of its nodes are disconnected from the parent Fabric network.

In general, the set of PoA authorities is semi-permanent: addition of a new authority requires a consensus between existing authorities (such a consensus does not have to be defined as 100 %, but changes of authority sets are comparably unfrequent and a delay until all authorities are live again can be acceptable). Likewise, removal of an authority requires a consensus. Obviously, it is rather risky to require a 100 % consensus before removing an authority, as a reluctant and non-cooperating authority would simply veto a removal.

B. PoA in Parity Ethereum

PoA restricts the block creation *capability* to a well-defined subset of the network nodes. For Ethereum, PoA is currently only provided by the Parity [27] implementation of Ethereum; we do not consider PoA for sidechains because these do not provide consensus participation to the network. Parity is not the implementation used for the Ethereum mainnet, and thus PoA is currently limited to private or consortial networks.

The Parity implementation is neither strictly permissioned nor strictly unpermissioned: new participants (Ethereum addresses) and new Ethereum nodes can come and go (unpermissioned), but the set of authorities is handled in a permissioned way.

The Parity-provided PoA is based on the Aura algorithm [28], which stands for Authority Round. An Aura authority (also called a validator) is an Ethereum address (i.e. a keypair) and the owner of the keypair can run several nodes, even though this does not influence the authority's chances, standing or earning. When a Parity Ethereum network with Aura is set up, the same (!) bootstrapping configuration has to be passed to all nodes, as well to nodes joining later. This configuration contains the set of authorities (i.e. their public addresses), and has to be agreed upon "out-of-band".

In Aura, only one authority confirms a given block and thus any given transaction. Aura devises that a validator should not confirm two consecutive blocks (although this is technically possible). With respect to network partitioning detection and prevention, the design of Aura makes no statements and live tests would have to be carried out as we could not find any testimonials of Aura's behaviour during network partitioning.

However, when the validator-in-charge is not acting (i.e. there is no node running with the identity of the validator-in-charge), it is automatically skipped once a certain time interval has passed, and the next validator becomes responsible. Consequently, we believe that in two independent (disconnected) network partition, two "validator rings" can emerge and thus, concurrent truths can be established. Since Aura cannot be configured to follow a policy such as "every validator must see m out of a validators" (so that one could impose $2 \times m > a$), it cannot be configured to *resist* network partitioning.

Additionally, a Parity node using Aura might detect that a certain validator is missing, but it cannot decide whether that validator is absent (dormant/shut down), which is a permitted situation, or whether it is active in a different partition. Therefore, the Aura implementation of PoA within Parity Ethereum appears unable to *detect* network partitioning.

C. Proof-of-Work

For PoW, the traditional consensus implementation (as exemplified by Ethereum or Bitcoin) is geared towards rewarding the fastest miner. In Ethereum, the many other competitors with correct but later solutions may only obtain "uncle rewards". The "work" in "Proof-of-Work" indicates that the miner has to solve a complex computational task in the process of creating a block from one or several transaction proposals. Strictly speaking, PoW as a pattern does not impose any approach on how the network participants agree on a mined block. Each serious blockchain implementation includes the mechanisms to check the cryptographic aspects of the mined block, but also to check the transaction contents of a block (e.g. to prevent "transaction injection" by the miner).

With regard to network partitioning, PoW is vulnerable - especially when combined with unpermissioned network design and probabilistic/gossip-based propagation. In many (if not most) PoW-using implemenations, there is absolutely no consensus-level mechanism to ensure that a block reaches *all* active network nodes, or even a given subset thereof. The probabilistic, "best-effort" approach of such implementations as Ethereum also does not include mechanisms to reconcile separate truths after a fully partitioned network is reunited. In Ethereum, it is possible to have branches in addition to the trunk's linear sequence of blocks (the trunk is defined as the sequence of blocks with the highest accumulated PoW, starting from the genesis block).

It is also permissible to append blocks to a branch, not only to the trunk; branches can be created when a minority accepts the branch as there is no fixed or proportional minimum amount of votes to confirm a block. The Ethereum yellowpaper (which defines the technology) does not impose any checks to ensure freedom of data conflicts across branches. In particular, there are "role switches" when a branch achieves the weight/length of a trunk and overtakes it in the next step, becoming a trunk itself. Thus, for PoW-using solutions, such problems make it unfeasible to hope for a built-in consensus-solution to overcome the network partitioning risks. This statement holds not just for permissionless (and inherently asynchronous) PoW implementations, but also for those permissioned PoW implementations which do not provide mechanisms to maintain integrity of the network. In other words, detection and prevention of network partitioning when using PoW must be performed on network level rather than on consensus level.

D. Proof-of-Stake and Proof-of-Space

We have left a detailed, implementation-specific analysis of Proof-of-Stake and Proof-of-Space to future work, but some preliminary considerations can already be made at this stage. PoS repeats (sometimes even for each block) the selection of a *single* node which "mines" the block; the selection algorithms vary and PoS is used in both permissioned and permissionless implementations. In the case of network partitioning, the miner selection step can run independently in each partition and block creation can also run in isolation. Thus, it is possible that conflicting truths emerge in each isolated partition. Likewise, Proof-of-Space suffers (at the consensus level) from the same potentional vulnerability as PoW and PoS, when a single miner is sufficient to create a new block. However, network-level partition detection and prevention can be implemented at the layers below PoS consensus algorithms.

E. The Perspective of a Node Client onto Network Partitioning

So far, we have described how network-participating nodes are involved in partitioning and the prevention of it. However, in a blockchain-based landscape, not every participating computer is a node with a replica of the ledger state. Node *clients* which access blockchain network nodes carry the ability to work with more than just one node - and more than just one node at a time. If the blockchain network is permissioned and based on PoA, it is possible to implement an approximative partitioning detection from the "node client" perspective.

From the "node client" perspective, the core question w.r.t. network partitioning is: how many nodes does the client know about, and how many of them have to supply information so that the "node client" trusts and accepts it? Most client application architectures live the "one answer is enough" pattern, while high-security applications use "n-version programming" (see e.g. [29]) to go beyond that. Working with multiple sources of information is also done in the sensor fusion and can be supported by fuzzy logic.

Even if the client is only enabled to contact a single node for a given single request, an improvement is possible by diversifying the request destinations and repeating the same request to a different destination. The underlying "multi-contact" pattern is similar to load balancing where the request routing can be the responsibility of the "node client", or there can be an additional and separate implementation to distribute the requests: the so-called load balancer. Obviously, the latter separation requires that the load balancer is highly available - there can be several balancers which a client knows about and (many more) workers which are not visible to the client; the load balancer distributes the client's requests.

IV. Conclusions and Future Work

Network partitioning is a well-known vulnerability which is relevant for distributed and peer-to-peer networks, such as those used for distributed ledgers and blockchains. Algorithms for partitioning prevention have been developed, and users expect to find partitioning prevention in blockchains and DLTs. In this paper, we have laid initial foundations for partitioning analysis in the context of blockchain, and have analyzed the role of consensus algorithms in this context. In particular, based on the analysis of Hyperledger Fabric, we have derived a formula to configure Proof-of-Authority consensus to avoid the creation of concurrent truths during accidental network partitioning.

For future work, we plan to extend our analysis to the situations with malicious partitioning, e.g. where a "Byzantine general" generates two conflicting blocks in two partitions. We also plan to analyze specific enterprise-grade products (such as Algorand, Hashgraph, Quorum, R3 Corda and others) and to perform deeper research in further consensus algorithms (such as ProgPoW, Paxos, RAFTand others) in more detail. Finally, we strive to devise executable and repeatable test procedures for live analysis of network partitioning scenarios, building on existing peer-to-peer network research.

Acknowledgements

Patrick Charrier, Sebastian Kemper, Robin Klemens, Peter Mahlmann, Artur Philipp, Malte Plescher, Michael Schnabel and Marius van der Wijden provided very helpful review findings.

References

[1] Z. Zheng, S. Xie, H. Dai, X. Chen, and H. Wang, "An Overview of Blockchain Technology: Architecture, Consensus, and Future Trends," in *2017 IEEE Intl. Congress on Big Data*, June 2017, pp. 557–564.
[2] E. Brewer, "CAP twelve years later: How the "rules" have changed," *Computer*, vol. 45, no. 2, pp. 23–29, Feb 2012.
[3] S. De Angelis, L. Aniello, R. Baldoni, F. Lombardi, A. Margheri, and V. Sassone, "PBFT vs Proof-of-Authority: Applying the CAP Theorem to Permissioned Blockchain," 2018.
[4] K. Zhang and H. Jacobsen, "Towards Dependable, Scalable, and Pervasive Distributed Ledgers with Blockchains," in *ICDCS 2018*, July 2018, pp. 1337–1346.
[5] J. A. Tran, G. S. Ramachandran, P. M. Shah, C. B. Danilov, R. A. Santiago, and B. Krishnamachari, "SwarmDAG: A Partition Tolerant Distributed Ledger Protocol for Swarm Robotics," *Ledger*, vol. 4, 2019.
[6] N. Kannengießer, S. Lins, T. Dehling, and A. Sunyaev, "Mind the Gap: Trade-Offs between Distributed Ledger Technology Characteristics," *CoRR*, 2019. [Online]. Available: http://arxiv.org/abs/1906.00861
[7] S. Jajodia, "Managing replicated files in partitioned distributed database systems," in *1987 IEEE Third International Conference on Data Engineering*, Feb 1987, pp. 412–418.
[8] M. J. Fischer, N. A. Lynch, and M. S. Paterson, "Impossibility of distributed consensus with one faulty process." Massachusetts Inst. of Techn., Cambridge lab for Computer Science, Tech. Rep., 1982.
[9] P. Ekparinya, V. Gramoli, and G. Jourjon, "The Attack of the Clones against Proof-of-Authority," September 2019. [Online]. Available: http://arxiv.org/abs/1902.10244
[10] P. Singhal and S. Masih, "MetaAnalysis of Methods for Scaling Blockchain Technology for Automotive Uses," 2019. [Online]. Available: http://arxiv.org/abs/1907.02602
[11] P. Ekparinya, V. Gramoli, and G. Jourjon, "Double-Spending Risk Quantification in Private, Consortium and Public Ethereum Blockchains," May 2018. [Online]. Available: https://arxiv.org/abs/1805.05004
[12] M. Dalui, B. Chakraborty, and B. K. Sikdar, "Quick Consensus Through Early Disposal of Faulty Processes," in *2009 IEEE International Conference on Systems, Man and Cybernetics*, Oct 2009, pp. 1989–1994.
[13] D. Buntinas, "Scalable Distributed Consensus to Support MPI Fault Tolerance," in *2012 IEEE 26th International Parallel and Distributed Processing Symposium*, May 2012, pp. 1240–1249.
[14] K. Saito and H. Yamada, "What's So Different about Blockchain? — Blockchain is a Probabilistic State Machine," in *ICDCSW 2016*, June 2016, pp. 168–175.
[15] A. Anjum, M. Sporny, and A. Sill, "Blockchain Standards for Compliance and Trust," *IEEE Cloud Computing*, vol. 4, no. 4, pp. 84–90, July 2017.
[16] X. Wang, X. Zha, W. Ni, R. P. Liu, Y. J. Guo, X. Niu, and K. Zheng, "Survey on blockchain for Internet of Things," *Computer Communications*, vol. 136, pp. 10 – 29, 2019. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0140366418306881
[17] Y. Hu, A. Manzoor, P. Ekparinya, M. Liyanage, K. Thilakarathna, G. Jourjon, and A. Seneviratne, "A Delay-Tolerant Payment Scheme Based on the Ethereum Blockchain," *IEEE Access*, vol. 7, pp. 33 159–33 172, 2019.
[18] T.-H. H. Chan, R. Pass, and E. Shi, "PaLa: A Simple Partially Synchronous Blockchain," *IACR Cryptology ePrint Archive*, vol. 2018, p. 981, 2018. [Online]. Available: https://eprint.iacr.org/2018/981.pdf
[19] I. Homoliak, S. Venugopalan, Q. Hum, and P. Szalachowski, "A Security Reference Architecture for Blockchains," April 2019. [Online]. Available: https://arxiv.org/abs/1904.06898
[20] S. Keshav, W. M. Golab, B. Wong, S. Rizvi, and S. Gorbunov, "RCanopus: Making Canopus Resilient to Failures and Byzantine Faults," October 2018. [Online]. Available: http://arxiv.org/abs/1810.09300
[21] W. Li, S. Andreina, J.-M. Bohli, and G. Karame, "Securing Proof-of-Stake Blockchain Protocols," in *DPM 2017*, Sept. 2017, pp. 297–315.
[22] H. Yu, I. Nikolic, R. Hou, and P. Saxena, "OHIE: Blockchain Scaling Made Simple," Nov. 2018. [Online]. Available: http://arxiv.org/abs/1811.12628
[23] P. Mahlmann and C. Schindelhauer, "Peer-to-Peer Networks based on Random Transformations of Connected Regular Undirected Graphs," in *SPAA 2005*, pp. 155–164.
[24] R. Pallavi and G. C. B. Prakash, "A review on network partitioning in wireless sensor and actor networks," in *iCATccT 2015*, Oct 2015.
[25] A. Baliga, "Understanding Blockchain Consensus Models," in *Persistent.com Whitepaper*, 2017.

[26] C. Cachin and M. Vukolic, "Blockchain Consensus Protocols in the Wild," 2017. [Online]. Available: http://arxiv.org/abs/1707.01873
[27] Proof-of-Authority Chains - Parity Tech Documentation (Wiki). [Online]. Available: https://wiki.parity.io/Proof-of-Authority-Chains
[28] Aura - Authority Round (Wiki Parity Tech Documentation). [Online]. Available: https://wiki.parity.io/Aura
[29] A. Avizienis, "The methodology of n-version programming," *Software fault tolerance*, vol. 3, pp. 23–46, 1995.