Establishing Standards for Consensus on Blockchains

Derek Sorensen

Pyrofex Corporation
derek@pyrofex.net

Abstract. We survey six popular blockchain consensus algorithms, showing that the blockchain community does not have well-established theoretical foundations. In this paper, we consolidate and unify these foundational notions, establishing high quality standards and exploring the comparative relationship between foundations used on these different algorithms. The framework established here is meant to be used by academic and industrial blockchain researchers as a foundation for the theory of consensus.

Keywords: Blockchain, Consensus algorithm, Nakamoto consensus, Byzantine fault tolerant, Network model, Adversarial model, Safety, Liveness, Finality

1 Introduction

Blockchain technology, as an academic discipline, is very much in its development phases. New journals, such as *Ledger* and *International Journal of Blockchains and Cryptocurrencies*, have emerged to make a space for academics to discuss the economic, legal, social, mathematical, and computer scientific implications of these technologies. To an academic mathematician entering the field, it is immediately obvious that blockchain-related consensus algorithms are in urgent need of both formalization and standardization. Since most blockchain consensus algorithms come out of industry, these almost never make it through a peer-review process. Instead, they appear on company websites, as white papers, or simply on the ArXiv.

We see important drawbacks from this phenomenon. The first is that blockchain scientists in general do not communicate with each other. While it can, admittedly, be difficult to have research-sensitive conversations because of economic interests, this phenomenon inhibits the field from achieving unity because observations, definitions, techniques, and results end up being rediscovered by each respective party. Not only does this waste time, but each party introduces nuances, and thus more confusion. The second, perhaps a symptom of the first, is that most blockchain consensus algorithms use their own seemingly *ad hoc* definitions of consensus, safety, liveness, finality, and their own, nuanced network assumptions. At times these call distinct notions by the same name, or equivalent notions by distinct names. For a scientist trying to compare blockchain

consensus algorithms, this lack of standardization and communication causes undue confusion and wastes significant amounts of time in comparing author's ideas regarding these more basic notions in addition to the algorithms.

This paper seeks to fulfill two purposes. The first is to compare and contrast the definitions, models, and network assumptions from several popular blockchain consensus algorithms. We show how the notions relate and differ to give motivation for our standardization and insights as to why comparison is so difficult and nearly always unfruitful. The second is to standardize these definitions and notions, relating them back to the foundations of classical, pre-Nakamoto literature on consensus. Our purpose here is not to give a survey of consensus, nor to give readers a sense of existing consensus algorithms. Rather, our purpose is to inform researchers who are either selecting a consensus algorithm for a new blockchain or building a consensus algorithm of their own, what high-quality foundations look like and why they must take them seriously.

In this paper, we explore six fundamental notions underlying consensus on a blockchain, which are: Byzantine agreement, network assumptions, the adversarial model, finality, safety, and liveness. For each of these notions, we demonstrate the lack of unity and the ensuing problems with the following consensus algorithms: Nakamoto's consensus algorithm [16] run on BTC and ETH, PBFT [4], Gosig [13], Best of Both Worlds (BoBW) [14], Ouroboros [12], and Honey Badger BFT [15]. Realizing that we could not survey every popular consensus algorithm, we chose to only survey from those that have published papers and that claim to have proofs of both safety and liveness in the face of a Byzantine adversary. Thus we neglect some popular algorithms such as Raft [6], which explicitly only treats crash faults, and Tangaroa [5], which lacks a proofs altogether. As part of this analysis we also propose standard definitions sufficiently general to apply to any consensus algorithm on a blockchain or blockdag.

2 Related Work

There have been several efforts to give summaries, descriptions, or taxonomies on different aspects of blockchain technology. While these are, in general, very useful, each has their own goals, and none of these work exclusively within the mathematical foundations of consensus like we do here.

For example, [23], [17], and [24] give surveys of consensus algorithms as they exist within the blockchain world. These introduce concepts such as proof of work or proof of stake, and give examples. We assume that our reader is familiar with consensus algorithms used within industry. Instead of giving a survey of the algorithms themselves, we survey their theoretical foundations.

Similarly, [20], [22], and [21] give surveys of consensus algorithms but do so from a practical perspective, describing the technology itself or its performance metrics. For example, [20] focuses on and analyzes the Stellar Consensus Protocol and the Linux Foundation's hyperledger product; [22] focuses on the issue of scaling, contrasting proof-of-work algorithms with BFT replication; and [21] gives an introduction to the technology surrounding Bitcoin, including con-

cepts such as double spending, transaction malleability (finality), mining, bonding/unbonding, traffic analysis, and so on. The features of different consensus algorithms mentioned in these articles, while important, do not strongly relate to the topic that we treat of consistency in mathematical models, foundations, and assumptions.

In [25], the authors propose a taxonomy with which one can understand and compare consensus algorithms. In some ways, the purpose of this paper is similar to ours. The standards we establish are meant to facilitate comparison. However, we do so from the standpoint of fundamental, mathematical assumptions, while [25] does so from a feature-oriented perspective. Furthermore, we seek to *set* mathematical standards that blockchain scientists will adopt so that the consensus algorithms produced within industry can be rationally compared to one another. These standards do not yet exist, and without them it is extremely difficult to perform any meaningful, mathematical comparison.

Perhaps the article closest in purpose and content to ours is [3], which explores blockchain consensus algorithms "in the wild." The authors briefly touch on definitions and assumptions related to safety, though they do not mention in rigorous detail the other key concepts that we cover such as network assumptions, adversarial model, liveness, and finality. Furthermore, the definition given for safety does not apply to consensus on blockdags such as [1] [2], [8], and [19]. While they do criticize some consensus algorithms, such as Tangaroa [5], as lacking in theoretical fondations, the authors neither propose standards nor thoroughly explore the essential foundations of consensus.

The mathematical standards we propose tie consensus algorithms that have emerged for the specific purpose of the blockchain to classical consensus algorithms made for general distributed systems. In the process, we uniquely give a survey of mathematical foundations to show the need for such standards to exist. Our principal contribution is to propose standards that ground blockchain consensus firmly in solid mathematical theory, making it more feasible to compare consensus algorithms in terms of their network and adversarial models, finality properties, and conditions for safety and liveness.

3 Byzantine Agreement

Every consensus algorithm for a practical blockchain tries to achieve *Byzantine agreement*, which is to achieve safety, liveness, and finality under some specified network assumptions and a given adversarial model. Byzantine agreement is the core problem to be solved, and so should have a standard, constant definition that we can all draw from. Surprisingly, only two of the algorithms we surveyed explicitly stated a definition of Byzantine agreement, and those definitions that were explicitly stated differed in nontrivial ways. The remaining algorithms neglected to mention the problem in any formal way. We find this perplexing, as one cannot know if a problem has been solved if there is no good definition to begin with.

To illustrate our point, consider those two algorithms, BoBW[14] and Honey Badger BFT[15], that actually stated definitions of Byzantine agreement. The former, in Definition 3.1, defines Byzantine agreement to be a distributed protocol Π that satisfies *validity*, *consistency*, and *p-termination*. The latter, in Appendix C, requires *agreement*, *termination*, and *validity*. In these definitions, BoBW's consistency corresponds to Honey Bader's agreement, but both definitions of termination and validity differ. Honey Badger's definition of termination corresponds to a quasi-combination of BoBW's validity and p-termination, without reference to any probability p. The notion of validity as defined in Honey Badger isn't even found in BoBW's definitions.

The importance of having a clear objective in research cannot be understated. We cannot achieve a goal we do not set. Even more troubling than inconsistent definitions is no definition at all. Not only does this betray a lack of rigorous thinking, but it can be actively deceptive. Since there was no rigorous problem definition to begin with, these papers usually declare their algorithms to be satisfactory if they meet some benchmark of safety and liveness that the authors place, seemingly arbitrarily. From these kinds of arguments, it is never clear if a given algorithm actually satisfies its purpose, since there was no concrete purpose to begin with.

We reiterate that the problem of achieving Byzantine agreement on a blockchain is simply to achieve safety, liveness, and finality under specified network assumptions and a adversarial model. These five mentioned components, then, must necessarily be standard notions if we are to claim to be after the same problem.

In all of the definitions that follow, we follow convention and use N to refer to the total number of members of consensus, and f to be the number of Byzantine nodes, those nodes whose behavior can be arbitrary. As these notions do not apply to Nakamoto consensus [16], where appropriate we specify the analogous notions specific to that case. We call members of consensus *validators*.

4 Network Assumptions

We prefer a minimalist approach to network assumptions. While it is impossible to guarantee consensus on a fully asynchronous network, it is possible to achieve consensus under some mild assumptions [10]. The weakest of these, which we see as the gold standard, is *intermittent synchrony*. A network is intermittently synchronous, or Δ-intermittently synchronous, if any interval of time can be extended to one during which the average message delay is at most Δ (for a more technical definition, see the Appendix of [15]).

Of the consensus algorithms we review in this paper, only Honey Badger has proofs based on intermittent synchrony. The rest assume variants or special cases of a slightly stronger assumption called *partial synchrony*. A network is partially synchronous if there exists some finite time bound Δ before which all messages get delivered. Researchers should be careful not to require knowledge, or even an estimate, of Δ at any point in the consensus algorithm. Most commonly,

safety is achievable in complete asynchrony; it is usually liveness that requires intermittent or partial synchrony. Partial synchrony is strictly stronger than intermittent synchrony [15], though as we remark at the end of this section, in practice the difference between the two is insignificant.

Many of the algorithms we compare make this partial synchrony assumption, some opting for a variation that says partial synchrony holds after some time t, often called the *Global Stabilization Time*. Most, however, make other subtle assumptions about the network that are difficult to compare with each other. For example, for safety Gosig assumes partial synchrony but also requires that any adaptive attacks on validators take effect only after a delay of time Δ. For liveness, it assumes partially synchronized clocks, meaning that any two validator's local clocks only differ by some fixed number [13, §3.2]. In contrast, in addition to assuming partial synchrony, Best of Both Worlds (BoBW) assumes that validators are in *lock step*, which means that they proceed in rounds of fixed time and any two validators are always within some fixed bound of each other in consensus. This may seem stronger than Gosig's partially synchronized clocks assumption, but as the authors of BoBW point out, one can achieve lockstep synchrony with partial synchrony and partially synchronized clocks, which they call *bounded drift* [14, p. 7].

Thus, aside from BoBW assuming that validators proceed through consensus in rounds of fixed length and Gosig assuming that adaptive attacks take effect after a delay, BoBW and Gosig carry the same network assumptions but present them with different terminology and in entirely different ways. For a researcher seeking to read and compare these algorithms, it is not easy to tell that they rest on almost identical network assumptions. Even with this insight in place, because of these additional, nuanced assumptions that BoBW and Gosig, respectively, make, it is yet unclear how to compare the algorithms because of a differing network model. In particular, it is unclear that either of these assumptions is easy to satisfy in practice, or even that one is stronger or less desireable than the other.

PBFT also assumes partial synchrony, but uses entirely different language than BoBW and Gosig to do it. Rather than assuming a bounded delay, the authors define partial synchrony to be that $delay(t)$, the time a message sent at time t is delivered to its recipient, cannot grow faster than t indefinitely[4, §3]. In other words, there is an upper bound Δ on the delay between a message being sent and it being received. Of the algorithms we surveyed, PBFT is the only one to make the partial synchrony assumption and nothing else. Still, it is difficult to compare its network model with others if such a fundamental notion is defined in this uncommon way.

Finally, Ouroboros' assumptions are simply incomparable to the others. It assumes a form of lockstep, without using any standard terminology, which appears to be strictly stronger than that of BoBW, as it says that any difference in validator's local clocks is insignificant in comparison to the length of computational steps in the algorithm. It also assumes, again without standard terminology, a very strong form of synchrony, which is that any message is received within

the length of a computational step in the algorithm. Recognizing how practically infeasible this is, Ouroboros is manifestly inferior to the others because these assumptions sharply curb the speed at which the network can execute the algorithm.

As the authors of Honey Badger point out, intermittent synchrony more accurately models the real world and is evidently superior to partial synchrony [15, §3]. We take the position that any researcher building a consensus algorithm should do so assuming only intermittent synchrony. While inferior, we also recognize that partial synchrony, in either form of the definition, still works in practice. If the algorithm requires assumptions additional to partial synchrony, the authors should provide evidence that they are readily satisfiable in practice for any network or algorithm. They should also try to quantify any resulting effects of these assumptions on network speed or algorithm performance. In order to make fruitful comparisons between consensus algorithms, authors should be explicit on any network assumptions that do not fall in either of these categories, including restrictions on hardware, requirements such as bandwidth or computation power, and so on.

Remark 1. The authors of Honey Badger mention that those consensus algorithms that require partial, as opposed to intermittent, synchrony often rely on the upper bound Δ in such a way that an intermittently synchronous network could starve the system. The example they give is a network that exhibits longer and longer periods of asynchrony, with corresponding longer and longer periods of synchrony. In the case of PBFT, this can starve the system, while Honey Badger would still make progress. However, for practical purposes this network would also starve Honey Badger if, for example, the network delays go on for more than a day, or a week, or a year.

The conclusion from their analysis is that an algorithm that performs well on an intermittently synchronous network can perform better in general than one that uses the partial synchrony assumption to deal with faults via timeouts. It is yet unclear how these algorithms that use the partial synchrony assumption for something other than timeouts compare in performance to those that can use intermittent synchrony. Since, in practice, messages still need to arrive in a reasonable time frame for any algorithm to be live, we do not see a meaningful difference between these intermittent and partial synchrony.

Standard Network Assumption	Intermittent synchrony, or Δ-intermittent synchrony, that one can always find an interval of time during which the average message delay is at most Δ.		
Algorithm	**Network Assumptions**	**Special Case of Standard?**	**How It Differs From The Standard**
Nakamoto	No network assumptions.		
PBFT	Assumes partial synchrony, using a unique definition, asserting that the function $delay(t)$ does not grow faster than t indefinitely.	Yes.	Using the function $delay(t)$ is not standard, but turns out to be a special case of partial synchrony. For a proof, see Proposition 4 in the Appendix.
Gosig	Assumes for safety partial synchrony and that adaptive attacks take effect only after a delay of time Δ. For liveness, also assumes partially synchronized clocks.	Yes.	In addition to partial synchrony, assumes a delay of time Δ in adaptive attacks, as well as partially synchronized clocks.
BoBW	Assumes partial synchrony as well as lock step synchrony.	Yes.	Assumes that validators proceed in rounds of fixed time, and that any two validator are always within some fixed bound of each other in consensus. The latter assumption is equivalent to partially synchronized clocks.
Ouroboros	Assumes, with nonstandard terminology, a form of lock step. From the definition it is unclear how that compares with other definitions of lock step. Also assumes a strong form of synchrony.	Yes.	Strengthens partial synchrony to strong synchrony. Also assumes a form of lock step synchrony.
Honey Badger	Assumes Δ-intermittent synchrony.	Yes.	This is the standard, of which partial synchrony is a special case.

Fig. 1. Network assumptions display little to no unity across different algorithms and are thus inherently difficult, if not impossible, to effectively compare with each other. In this table, we compare each to the standard network assumption. As indicated, our recommended standard assumption is weaker than each of those that had explicit assumptions, and strictly so than all but Honey Badger's. This means that each of the network assumptions we analyzed are a special case of the standard, Δ-intermittent synchrony.

5 Adversarial Model

In general we see it as essential that researchers assume the strongest adversarial model possible, so as to give the most accurate picture of an algorithm's strength. This includes assumptions about the proportion of malicious agents and their behavior.

The standard is to assume that some number less than $\frac{N}{3}$ of validators are Byzantine. Their behavior can be arbitrary, including intentionally malicious activity, collusion, or inactivity. If the protocol uses cryptographic tools, there should be clear assumptions about what a malicious validator can and cannot do. In this case it is reasonable to assume that Byzantine validators can't forge signatures or invert hash functions. It is, however, perfectly possible that all the Byzantine nodes collude, even controlling the network to slow consensus, but not to violate network assumptions. For this reason, again, network assumptions must be as light as possible.

Researchers proposing a new consensus algorithm should assume the maximum number of Byzantine validators, $\lfloor \frac{N-1}{3} \rfloor$. These malicious nodes should be assumed to be both colluding and controlling the network, where adversaries can delay messages, duplicate them, or deliver them out of order. Any restrictions on the adversary should be explicitly stated. These include restrictions on which messages the network can drop, what a Byzantine node may or may not know, and even assumptions relating to cryptographic techniques such as hashes or signatures. For example, PBFT states,

> We do assume that the adversary cannot delay correct nodes indefinitely. We also assume that the adversary (and the faulty nodes it controls) are computationally bound so that (with very high probability) it is unable to subvert the cryptographic techniques mentioned above. [4, §2]

BoBW is similarly high quality, in that it assumes a fully adaptive, knowledgeable adversary that can know the entire internal states of all other Byzantine validators. As part of their network assumptions, the authors state,

> The adversary has full control over the network: It has the power to delay messages arbitrarily up to Δ time steps. It can reorder messages, and it can make some messages arrive multiple times at its intended recipient. [14, §3]

Nakamoto's adversarial model is simple, stating that "the system is secure as long as honest nodes collectively control more CPU power than any cooperating group of attacker nodes." [16, §1]. Assumptions related to the hash function, *etc.*, are implicit in the fact that CPU power is the metric of adversarial power.

Honey Badger, on the other hand, only explicitly states that "the [message] delivery schedule is entirely determined by the adversary, but every message sent between correct nodes must eventually be delivered"[15, §4.1]. It reasons about the adversary throughout the paper, but rather than giving a specific model of adversarial behavior beforehand, one discovers the assumed adversarial abilities as the adversary appears in arguments.

There is, for the most part, unity in the blockchain literature about the behavior of Byzantine validators. We advocate explicit, technical definitions of possible Byzantine behavior in order to provide as much rigor as possible. In general, Byzantine behavior should be assumed to be *truly arbitrary* unless the algorithm explicitly states limitations. Any limitations imposed should be clearly stated, though we stress that limitations on Byzantine behavior make comparative analysis difficult at best.

6 Finality

The highest quality consensus algorithms for the blockchain must have both a precise measure of finality and a point at which a block or transaction can be declared final such that, under the network and adversarial assumptions, it is impossible (not just improbable) to be rewritten. Of course, this notion of finality must depend on the assumptions about the network and adversarial model being satisfied. Researchers should also explore any inherent limitations on latency and throughput that the finality conditions in their algorithms impose.

Perhaps unsurprisingly, most algorithms do not mention finality explicitly, but many do exhibit good finality properties. In particular, blockchains built on a consensus algorithm based on rounds of message passing, including most of those we survey here, have an inherent notion of finality because the network decides on one block at a time. Thus once a block has passed through consensus, it becomes final.

For these algorithms, finality on a chain is straightforward, but finality on, for example, a DAG, would be less straightforward and should come with proofs. Other situations which complicate finality are those, like that of Nakamoto's algorithm, that can only give a probability of finality that approaches 1 as time goes on. For both of these kinds of algorithms, researchers should devote the time to give a thorough exposition on their algorithm's finality properties.

The kind of probabilistic finality in Nakamoto's consensus algorithm seems inherent to unpermissioned chains, since one can never fully know the entire set of participants. Understandably, then, of the existing chains, only the permissioned chains give exact statements on finality. Considering the nontrivial security breaches in these unpermissioned chains, most recently ETC [11], we are partial to permissioned chains principally because of their finality properties. Despite having a proof of safety, if a consensus algorithm cannot give definitive finality it can, in practice, not always be safe if the heuristic used to declare a block "safe" is not always reliable.

In the following sections we treat safety and liveness using a notion of exact finality. This, of course, precludes straightforward compatibility with Nakamoto-style proof-of-work consensus. As we mention in Section 8, the definitions we give can still be used with probabilistic finality with some minor modifications.

7 Safety

Safety is the first of the two core results that any consensus algorithm absolutely must have. It inherently relies on the notions we have established above, namely network assumptions, adversarial model, and finality. As it is trivial to write an algorithm that is live but not safe, namely that in which each validator produces and finalizes its own blocks only, a consensus algorithm without a proof of safety is pragmatically worthless. In the definition and discussion that follows we refer to blocks as the subject of consensus; however, safety can be equivalently proved on transactions and smart contracts.

Definition 2. *A consensus algorithm is safe if and only if:*

1. *Every finalized block is valid, and*
2. *For any pair of honest validators v and v' and blocks b and b', if v considers b to be final and v' considers b' to be final, then b and b' do not conflict.*

This definition requires the notions of *validity* and *conflicts* between blocks, both of which should be explicitly defined. For a traditional blockchain, a block is valid if it is constructed correctly according to protocol. Most neglect to mention validity because it is implicit in the adversarial model. That is, since an adversary cannot forge signatures, it is easy to reliably identify a block's sender. The same is true of transactions from clients. As long as the block structure can be easily verified to be correct by looking at the contents, one generally need not prove validity. However, a good researcher will carefully consider the network and adversarial models to ensure that there are no issues of validity, supplying a proof if validity is not self-evident. For an excellent example of a definition of block validity, see Blockmania [8, §2].

Again, on a traditional blockchain, two blocks conflict if they have the same block height and are not equal. In other words, an algorithm is safe if it cannot fork. Most of the algorithms we surveyed capture the essence of safety as we define it here, but the definitions are not equivalent. Definitions of safety tend to come in two camps. The first is that if one honest validator commits a block, then all other honest validators will eventually do the same (see Ouroboros and Honey Badger). The second is that if any two honest validators compare their chains, their blocks at any given height will be identical (see Nakamoto consensus, PBFT, Gosig, and BoBW). While the first implies the second, the second does not imply the first, as the first has a sort of "totality" property (see Proposition 5 in the Appendix for a proof). We opt for the second of these two definitions for safety, as such a totality property decidedly relates to liveness, rather than safety.

Since it's easy to determine if two blocks conflict on a chain by checking block height, we might be tempted to define property (2) of Definition 2 as if honest validators v and v' each consider b and b' to be final, then either b and b' are at different block heights or they are equal. However, this excludes consensus algorithms that build a block*dag* as opposed to a blockchain, such as [1], [2], [8], and [19]. In this case, a block's height is defined to be one more than

Algorithm	Terminology for Safety	Translated
Nakamoto	Negligible probability of forking or of a block being overwritten.	No forking.
PBFT	Impossibility of forking.	No forking.
Gosig	Validity and consistency.	If two honest players committed blocks at height n, the blocks are equal.
BoBW	Validity and consistency	If two honest players committed blocks at height n, the blocks are equal.
Ouroboros	Persistence.	If an honest player commits a block, all honest players eventually will.
Honey Badger	Agreement and total order.	If an honest player commits a block, all honest players eventually will, and if two honest players committed blocks at height n, the blocks are equal.

Fig. 2. Modulo the mention of validity, each of the algorithms we surveyed defined safety as the property that forking is impossible, honest players have the same blocks at the same heights, or that any output from an honest player would eventually be output by all honest players. While these definitions are not actually equivalent, coupled with liveness properties they are (see Proposition 5 in the Appendix.Aside from that, the only inconsistency in definitions was a mention of validity, which did not always happen.

the maximum height of its parents, so equality at a given block height is not a sufficient notion to describe conflicts, since there can be many blocks at a any height.

Thus on a blockdag one must clearly define what it means for two blocks to conflict. In Casanova, for example, the authors clearly define the notion of a *conflict domain*, and two blocks conflict if and only if they belong to the same conflict domain [2]. In contrast, in Blockmania the notion of a block conflict is intimately related to the notion of block validity [8, §2], and in Prism conflicts relate to the idea of consistency [1, p.36], but neither give explicit definitions. For researchers building a consensus algorithm for a blockdag, we recommend clearly defining the notion of block conflicts in order to produce high quality, rigorous proofs.

8 Liveness

Liveness is the second core result of any consensus protocol. As it is trivial to produce a safe algorithm that is not live, where validators simply do nothing, an algorithm with all the above properties clearly defined, including safety, is worth almost nothing until one can prove liveness. Liveness, like safety, depends

on network assumptions, the adversarial model, and finality. It is crucial to clearly understand each of these as they relate to a consensus algorithm in order to give a proof of liveness.

Definition 3. *A consensus algorithm is live if and only if for every honest validator v and finite time t:*

1. *For any block b generated by v, there exists some time $t' > t$ where v will have finalized b, and*
2. *There is some time $t'' \geq t'$ where every other honest validator will have also finalized b.*

Both of the properties of liveness are important. The first says that correct blocks can make it through to finalization, while the second block says they do so transitively, *i.e.* once they have been finalized by one honest node, they will eventually be finalized by all honest nodes.[1] In certain situations, such as in algorithms on a traditional blockchain, where the network decides on one block at a time and only moves forward after a decision has been made, the second requirement can be implicitly satisfied by the structure of the system. Merely requiring termination suffices in that case because we know that consensus must begin, and that correct nodes must be consistent. Therefore, if it terminates, then it must necessarily satisfy the second requirement. In these situations, liveness can be simply referred to as *termination*.

Liveness in the algorithms we surveyed falls generally into two groups. The first is that honestly generated transactions eventually finalize (see Nakamoto consensus, Ouroboros, and Honey Badger). PBFT gives a similar, but nuanced, definition, that the network responds correctly to transactions. The second is that honest parties eventually terminate with some (valid) output. Interestingly, these two are not only not equivalent, but they are independent from each other (see Proposition 6 from the Appendix). Our definition combines and refines the concepts from both groups, which is significant because of their independence. Figure 3 gives more details.

It is worth noticing that our definition relies heavily on the notion of finality, and thus lends itself to permissioned chains better than unpermissioned ones. However, if, like in the case of BTC or ETH, finality is defined by waiting a fixed number of blocks, then this definition may still apply. Indeed, the paper that proved Nakamoto's algorithm to be safe and live uses the strictly stronger definition of liveness given both by Ouroboros and Honey Badger [18]. That is that an honest miner (or validator) that submits blocks to the blockchain will eventually get them committed. Our definition of liveness requires a specified moment of finality, thus the existence of t' and t''. A consensus algorithm on an unpermissioned network that, like Nakamoto's algorithm, can only give probabilistic guarantees of finality can modify our definition to talk about the limit as time goes to infinity instead of specific instants t' and t'', which may not exist.

[1] As noted in the last section, some definitions of safety incorporate the second property implicitly.

Algorithm	Definition of Liveness	Translated
Nakamoto	Honestly generated transactions eventually finalize.	Honestly generated transactions eventually finalize.
Ouroboros	Honestly generated transactions eventually finalize.	Honestly generated transactions eventually finalize.
Honey Badger	Censorship resilience.	Honestly generated transactions eventually finalize (if they're input to $N - f$ correct nodes).
PBFT	Clients eventually receive correct replies to their requests.	The network responds correctly to transactions.
BoBW	Honest parties eventually terminate with some output.	Honest parties eventually terminate with some output.
Gosig	Starting from any point in time, there is a finite waiting period before any given honest player will commit a valid block.	Honest parties eventually terminate with some (valid) output.

Fig. 3. Definitions of liveness come in two principal groups: that honestly generated transactions eventually finalize (or a variant of that), and that honest players eventually have an output. These two definitions are not equivalent (see Proposition 6 in the Appendix for details). The definition we supply combines the essential ideas of both groups.

While being unable to specify specific moments of finality is less than ideal, we consider this to be the standard for liveness on an unpermissioned chain.

9 Acknowledgment

Many thanks to my colleague Justin Meiners, who is always willing to dig into a good math problem.

10 Conclusion

The blockchain community, as it matures scientifically, must have firm theoretical foundations on which to build the theory of consensus on blockchains. Without standardized foundations, notation, terminology, and assumptions, it is difficult, if not impossible, to make fruitful comparisons between consensus algorithms for the blockchain. To this end, this paper surveys six popular consensus algorithms and, noting problematic inconsistencies, proposes a strong, standard, theoretical foundation on which any consensus algorithm should be able to draw. Technical rigor and standardization with regards to Byzantine consensus, network assumptions, the adversarial model, finality, safety, and liveness are necessary for confidence in the algorithms themselves. It is our hope that through these foundations we will be able to reason about and compare consensus algorithms more successfully, avoiding security problems overlooked either through casual reasoning or through problematic theoretical foundations.

11 Appendix

Proposition 4. *For any time t, let $delay(t)$ equal the delay a message sent at time t takes to be received. Then requiring that $delay(t)$ does not grow faster than t indefinitely is a special case of partial synchrony, but they are not equivalent.*

Proof. We first treat the case that time t is a real number in the interval $[0, \infty)$, where 0 is the time that the first message is sent. Thus $delay(t)$ must be at least piecewise differentiable to have the notion of "growth." Suppose that $delay(t)$ does not grow faster than t indefinitely. Since $delay(t)$ is piecewise differentiable on $[0, \infty)$, it is piecewise continuous on the same interval. Let G be the set of all $t' \in [0, \infty)$ such that the derivative of $delay(t)$ at t' is greater than 1 (in other words, $delay(t)$ is growing faster than t at t'). By assumption, G is bounded above, and therefore has a least upper bound. Let us call that t_u. Now consider $delay(t) - t$ on the interval $[0, t_u]$. Since $[0, t_u]$ is closed and bounded, it is compact. And since $delay(t) - t$ is piecewise continuous on $[0, t_u]$, $delay(t) - t$ achieves its maximum value Δ on $[0, t_u]$. Note that no message delay ever exceeds Δ, and thus we have partial synchrony.

Going the other direction, we see that there could be cases in which $delay(t)$ could grow faster than t indefinitely but still achieve partial synchrony. In particular, we still achieve partial synchrony any time the integral from 0 to ∞ of $(\frac{d}{dt}\ delay(t)) - 1$ converges. In this case, we have partial synchrony where Δ is the value of the integral.

In this proof we assumed $delay(t)$ to be piecewise differentiable because it has a notion of "growth" unspecified by the authors of PBFT. This treats the case that t is a real number in the interval $[0, \infty)$. If time is treated as discrete time slots, then the proofs only involve a finite number of points, and thus are trivial. □

Proposition 5. *Consider the following definitions of safety.*

1. *If any honest player commits a block, then all honest players eventually commit a block.*
2. *If two honest players have a block at height n, then their blocks are equal.*
3. *The chain will never fork.*

Then the first implies the second and third, the second and third are equivalent, and neither the second nor the third implies the first.

Proof. (1) $\implies$ (2) : Suppose that honest player v_1 has block b at height n, and honest player v_2 has block b' at height n. If $b \neq b'$, then eventually v_1 will commit b' at height n, which is a contradiction since v_1 is honest.

(2) $\iff$ (3) : Going forward, if the chain forks, then two honest players must have different blocks at the same height by definition. Going backward, if two honest players have different blocks at the same height, there is by definition a fork.

(2), (3) $\not\implies$ (1) : Finally, if v_1 has block b at height n, and v_2's chain is not yet at height n, if v_2 doesn't commit any more blocks to the blockchain, v_1 and v_2

can satisfy definition (2) but will never satisfy definition (1). If we have liveness, then v_2 will eventually commit b at height n. Thus every honest validator will eventually commit b at height n, and we have property (1). □

Proposition 6. *Consider the following definitions of liveness.*

1. *Clients eventually receive correct replies to their requests.*
2. *Honestly generated transactions eventually finalize.*
3. *(Censorship resilience) Honestly generated transactions submitted to $N - f$ correct nodes eventually finalize.*
4. *Honest nodes eventually produce output.*

Then (1) $\implies$ (2) $\implies$ (3)*, but nothing else. In particular, none of these definitions are equivalent and* (4) *is independent of the rest.*

Proof. (1) $\implies$ (2): If clients eventually receive correct replies to their requests, then they will receive confirmations of honestly generated transactions. The other direction is not true, as an algorithm could satisfy (2) while not responding to incorrect transactions.

(2) $\implies$ (3): If an honestly generated transaction always finalizes, then it will in the case that it's submitted to $N - f$ correct nodes. Going the other direction, there is an equivalence only if $N - f$ correct nodes have to see and approve a transaction in order to finalize one. Since there are algorithms that don't require that, the backward direction does not work.

(4) is independent: Clients could receive correct replies to their requests, and transactions can finalize, while an honest node is partitioned away or has temporarily crashed. Coupled with some totality property, an implication would hold from any of the first three definitions to the fourth. We can't go the other way to any of the three, since honest nodes producing an output does not necessarily end in finalization, *e.g.* if the outputs are different. Furthermore, if a client submits an honest transaction to a Byzantine node, the network could ignore the message, satisfy (4), and not satisfy the first three definitions.

References

1. Bagaria, V., Kannan, S., Tse, D., Fanti, G., Viswanath, P. *Deconstructing the Blockchain to Approach Physical Limits.* (No publisher) 2018 https://arxiv.org/abs/1810.08092
2. Butt, K., Sorensen, D., Stay, M. *Casanova.* No publisher (2018) https://arxiv.org/abs/1812.02232
3. Cachin, C., Vukolić "Blockchain Consensus Protocols in the Wild." No Publisher (2017) https://arxiv.org/pdf/1707.01873.pdf
4. Castro, M., Liskov, B. *Practical Byzantine Fault Tolerance.* Proceedings of the Third Symposium on Operating Systems Design and Implementation (1999) http://pmg.csail.mit.edu/papers/osdi99.pdf
5. Copeland, C., Zhong, Hongxia "Tangaroa: a Byzantine Fault Tolerant Raft." Class project in Distributed Systems, Stanford University (2014) http://www.scs.stanford.edu/14au-cs244b/labs/projects/copeland_zhong.pdf

6. Coreos, *Coreos Raft*. No publisher (2014) https://github.com/coreos/raft
7. Crain, T., Gramoli, V., Larrea, M., Raynal, M. *DBFT: Efficient Byzantine Consensus with a Weak Coordinator and its Application to Consortium Blockchains*. No publisher (2018) https://arxiv.org/pdf/1702.03068.pdf
8. Danezis, G., Hrycyszyn, D. *Blockmania: from Block DAGs to Consensus*. (No publisher) 2018 https://arxiv.org/abs/1809.01620
9. Croman, K., *et al. On Scaling Decentralized Blockchains (A Position Paper)*. International Financial Cryptography Association (2016) https://fc16.ifca.ai/bitcoin/papers/CDE+16.pdf
10. Fischer, M. J., Lynch, N. A., Paterson, M. S. "Impossibility of Distributed Consensus with One Faulty Process." *Journal of the Association for Computing Machinery* **32.2** 374-382 (1985) doi:10.1145/3149.214121
11. Girimath, A. *Ethereum Classic [ETC]: A deep-dive into 51% attack leading to the loss of $1.1 million worth in ETCs*. AmbCrypto (2019) https://ambcrypto.com/ethereum-classic-etc-a-deep-dive-into-51-attack-leading-to-the-loss-of-1-1-million-worth-etcs/
12. Kiayias, A., Russell, A., David, B., Oliynykov, R. *Ouroboros: A Provably Secure Proof-of-Stake Blockchain Protocol*. No publisher (2017) https://eprint.iacr.org/2016/889.pdf
13. Li, Peilun., Wang, Guosai., Chen, Xiaoqi., Xu, Wei. *Gosig: Scalable Byzantine Consensus on Adversarial Wide Area Network for Blockchains*. No publisher (2018) https://arxiv.org/pdf/1802.01315.pdf
14. Loss, J., Moran, T. *Combining Asynchronous and Synchronous Byzantine Agreement: The Best of Both Worlds*. Cryptology ePrint Archive (2018) https://eprint.iacr.org/2018/235.pdf
15. Miller, A., Xia, Yu., Croman, K., Shi, Elaine., Song, D. *The Honey Badger of BFT Protocols*. Cryptology ePrint Archive (2016) https://eprint.iacr.org/2016/199.pdf
16. Nakamoto, S. "Bitcoin: A Peer-to-Peer Electronic Cash System." No Publisher (2008) https://bitcoin.org/bitcoin.pdf
17. Nguyen, G., Kim, K., "A Survey about Consensus Algorithms Used in Blockchain", *J Inf Process Syst* 4(1):101-128, 2018.
18. Pass, R., Seeman, L., Shelat, A. *Analysis of the Blockchain Protocol in Asynchronous Networks*. No publisher (2016) https://eprint.iacr.org/2016/454.pdf
19. Popov, S. *The Tangle*. No publisher (2018) https://assets.ctfassets.net/r1dr6vzfxhev/2t4uxvsIqk0EUau6g2sw0g/45eae33637ca92f85dd9f4a3a218e1ec/iota1_4_3.pdf
20. Sankar, L. S., Sindhu, M., Sethumadhavan, M., "Survey of consensus protocols on blockchain applications", *2017 4th International Conference on Advanced Computing and Communication Systems (ICACCS)*, pp. 1-5, Jan 2017.
21. Tschorsch, F., Scheuermann, B., "Bitcoin and Beyond: A technical survey on decentralized digital currencies." *IEEE Communications Surveys & Tutorials*, 18(3):464, 2016.
22. Vukolić, M. "The Quest for Scalable Blockchain Fabric: Proof-of-Work vs. BFT Replication" *Lecture Notes in Computer Science* (2016): 112–125.
23. Wahab, A., Memood, W., "Survey of Consensus Protocols." (No publisher) 2018 https://arxiv.org/abs/1810.03357
24. Wang, W., et al. "A Survey on Consensus Mechanisms and Mining Strategy Management in Blockchain Networks." (No publisher) 2018 https://arxiv.org/abs/1805.02707
25. Xu, X., et al. "A Taxonomy of Blockchain-Based Systems for Architecture Design" *2017 IEEE International Conference on Software Architecture*