Byzantine Generals in the Permissionless Setting

Andrew Lewis-Pye[1(✉)] and Tim Roughgarden[2,3]

[1] London School of Economics, London, UK
a.lewis7@lse.ac.uk
[2] Columbia University, New York, USA
[3] a16z Crypto, New York, USA
troughgarden@a16z.com

Abstract. Consensus protocols have traditionally been studied in the *permissioned* setting, where all participants are known to each other from the start of the protocol execution. What differentiates the most prominent blockchain protocol Bitcoin [15] from these previously studied protocols is that it operates in a *permissionless* setting, i.e. it is a protocol for establishing consensus over an unknown network of participants that anybody can join, with as many identities as they like in any role. The arrival of this new form of protocol brings with it many questions. Beyond Bitcoin and other proof-of-work (PoW) protocols, what can we prove about permissionless protocols in a general sense? How does the recent stream of work on permissionless protocols relate to the well-developed history of research on permissioned protocols?

To help answer these questions, we describe a formal framework for the analysis of both permissioned and permissionless systems. Our framework allows for "apples-to-apples" comparisons between different categories of protocols and, in turn, the development of theory to formally discuss their relative merits. A major benefit of the framework is that it facilitates the application of a rich history of proofs and techniques for permissioned systems to problems in blockchain and the study of permissionless systems. Within our framework, we then address the questions above. We consider a programme of research that asks, "Under what adversarial conditions, and for what types of permissionless protocol, is consensus possible?" We prove several results for this programme, our main result being that *deterministic* consensus is not possible for permissionless protocols.

Keywords: Consensus · Proof-of-Work · Proof-of-Stake · Proof-of-Space

1 Introduction

The Byzantine Generals Problem [14,18] was introduced by Lamport, Shostak and Pease to formalise the problem of reaching consensus in a context where faulty processors may display arbitrary behaviour. The problem has subsequently become a central topic in distributed computing. Of particular relevance to us

F. Baldimtsi and C. Cachin (Eds.): FC 2023, LNCS 13950, pp. 21–37, 2024.
https://doi.org/10.1007/978-3-031-47754-6_2

here are the seminal works of Dwork, Lynch and Stockmeyer [8], who considered the problem in a range of synchronicity settings, and the result of Dolev and Strong [7] showing that, even in the strongly synchronous setting of reliable next-round message delivery with PKI, $f+1$ rounds of interaction are necessary to solve the problem if up to f parties are faulty.

The Permissionless Setting (and The Need for a Framework). This rich history of analysis considers the problem of consensus in the *permissioned* setting, where all participants are known to each other from the start of the protocol execution. More recently, however, there has been significant interest in a number of protocols, such as Bitcoin [15] and Ethereum [3], that operate in a fundamentally different way. What differentiates these new protocols is that they operate in a *permissionless* setting, i.e. these are protocols for establishing consensus over an unknown network of participants that anybody can join, with as many identities as they like in any role. Interest in these new protocols is such that, at the time of writing, Bitcoin has a market capitalisation of over $400 billion.[1] Given the level of investment, it seems important to put the study of permissionless protocols on a firm theoretical footing.

Since results for the permissioned setting rely on bounding the number of faulty participants, and since there may be an *unbounded* number of faulty participants in the permissionless setting, it is clear that classical results for the permissioned setting will not carry over to the permissionless setting directly. Consider the aforementioned proof of Dolev and Strong [7] that $f+1$ rounds are required if f many participants may be faulty, for example. If the number of faulty participants is unbounded, then the apparent conclusion is that consensus is not possible. To make consensus possible in the permissionless setting, some substantial changes to the setup assumptions are therefore required. Bitcoin approaches this issue by introducing the notion of 'proof-of-work' (PoW) and limiting the computational (or hashing) power of faulty participants. A number of papers [9,10,16] consider frameworks for the analysis of Bitcoin and other PoW protocols. The PoW mechanism used by Bitcoin is, however, just one approach to defining permissionless protocols. As has been well documented [2], proof-of-stake (PoS) protocols, such as Ouroboros [13] and Algorand [6], are a form of permissionless protocol with very different properties, and face a different set of design challenges. As we will expand on here, there are a number of reasons why PoS protocols do not fit into the previously mentioned frameworks for the analysis of Bitcoin. The deeper question remains, how best to understand permissionless protocols more generally?

Defining a Framework. Our first aim is to describe a framework that allows one to formally describe and analyse both permissioned and permissionless protocols in a general sense, and to compare their properties. To our knowledge, our framework is the first capable of modelling all significant features of PoW and PoS protocols simultaneously, as well as other approaches like proof-of-space [19].

[1] See www.coinmarketcap.com for a comprehensive list of cryptocurrencies and their market capitalisations.

This allows us to prove general impossibility results for permissionless protocols. The framework is constructed according to two related design principles:

1. Our aim is to establish a framework capable of dealing with permissionless protocols, but which is as similar as possible to the standard frameworks in distributed computing for dealing with permissioned protocols. As we will see in Sects. 3 and 4, a major benefit of this approach is that it facilitates the application of classical proofs and techniques in distributed computing to problems in 'blockchain' and the study of permissionless protocols.
2. We aim to produce a framework which is as accessible as possible for researchers in blockchain without a strong background in security. To do so, we blackbox the use of cryptographic methods where possible, and isolate a small number of properties for permissionless protocols that are the key factors in determining the performance guarantees that are possible for different types of protocol (such as availability and consistency in different synchronicity settings).

In Sect. 2 we describe a framework of this kind, according to which protocols run relative to a *resource pool*. This resource pool specifies a *resource balance* for each participant over the duration of the execution (such as hashrate or stake in the currency), which may be used in determining which participants are permitted to make broadcasts updating the state.

Byzantine Generals in the Permissionless Setting. Our second aim is to address a programme of research that looks to replicate for the permissionless setting what papers such as [7,8,14] achieved for the permissioned case. Our framework allows us to formalise the question, "Under what adversarial conditions, under what synchronicity assumptions, and for what types of permissionless protocol (proof-of-work/proof-of-stake/proof-of-space), are solutions to the Byzantine Generals Problem possible?" In fact, the theory of consensus for permissionless protocols is quite different than for the permissioned case. Our main theorem establishes one such major difference. All terms in the statement of Theorem 1 below will be formally defined in Sects. 2 and 3. Roughly, the adversary is q-bounded if it always has at most a q-fraction of the total resource balance (e.g. a q-fraction of the total hashrate).

Theorem 1. *Consider the synchronous and permissionless setting, and suppose* $q \in (0, 1]$. *There is no deterministic protocol that solves the Byzantine Generals Problem for a* q*-bounded adversary.*

The positive results that we previously mentioned for the permissioned case concerned deterministic protocols. So, Theorem 1 describes a fundamental difference in the theory for the permissioned and permissionless settings. With Theorem 1 in place, we then focus on probabilistic solutions to the Byzantine Generals Problem. We leave the details until Sects. 3 and 4, but highlight below another theorem of significant interest, which clearly separates the functionalities that can be achieved by PoW and PoS protocols.

Separating PoW and PoS Protocols. The resource pool will be defined as a function that allocates a resource balance to each participant, depending on time and on the messages broadcast by protocol participants. One of our major concerns is to understand how properties of the resource pool may influence the functionality of the resulting protocol. In Sects. 2, 3 and 4 we will be concerned, in particular, with the distinction between scenarios in which the resource pool is given as a protocol input, and scenarios where the resource pool is unknown. We refer to these as the *sized* and *unsized* settings, respectively. PoS protocols are best modelled in the sized setting, because the way in which a participant's resource balance depends on the set of broadcast messages (such as blocks of transactions) is given from the start of the protocol execution. PoW protocols, on the other hand, are best modelled in the unsized setting, because one does not know in advance how a participant's hashrate will vary over time. The fundamental result when communication is partially synchronous is that no PoW protocol gives a probabilistic solution to the Byzantine Generals Problem:

Theorem 3. *There is no permissionless protocol giving a probabilistic solution to the Byzantine Generals Problem in the unsized setting with partially synchronous communication.*

In some sense, Theorem 3 can be seen as an analogue of the CAP Theorem [1,11] for our framework, but with a trade-off now established between 'consistency' and weaker notion of 'availability' than considered in the CAP Theorem (and with the unsized setting playing a crucial role in establishing this tradeoff). For details see Sect. 4.

1.1 Related Work

In the interests of conserving space, we describe here the most relevant related papers and refer the reader to Appendix 1 for a more detailed account.[2]

The Bitcoin protocol was first described in 2008 [15]. Since then, a number of papers (see, for example, [10,12,16,17]) have considered frameworks for the analysis of PoW protocols. These papers generally work within the UC framework of Canetti [4], and make use of a random-oracle (RO) functionality to model PoW. As we shall see in Sect. 2, however, a more general form of oracle is required for modelling PoS and other forms of permissionless protocol. With a PoS protocol, for example, a participant's apparent stake (and their corresponding ability to update state) depends on the set of broadcast messages that have been received, and *may therefore appear different from the perspective of different participants* (i.e. unlike hashrate, measurement of a user's stake is user-relative). In Sect. 2 we will also describe various other modelling differences that are required to be able to properly analyse a range of attacks, such as 'nothing-at-stake' attacks, on PoS protocols.

In [9], the authors considered a framework with similarities to that considered here, in the sense that ability to broadcast is limited by access to a

[2] For appendices, see the arXiv version: https://arxiv.org/abs/2101.07095.

restricted resource. In particular, they abstract the core properties that the resource-restricting paradigm offers by means of a *functionality wrapper*, in the UC framework, which when applied to a standard point-to-point network restricts the ability to send new messages. However, the random oracle functionality they consider is appropriate for modelling PoW rather than PoS protocols, and does not reflect, for example, the sense in which resources such as stake can be user relative (as discussed above), as well as other significant features of PoS protocols discussed in Sect. 2.3.

In [20], a model is considered which carries out an analysis somewhat similar to that in [10], but which blackboxes all probabilistic elements of the process by which processors are selected to update state. Again, the model provides a potentially useful way to analyse PoW protocols, but does not reflect PoS protocols in certain fundamental regards. In particular, the model does not reflect the fact that stake is user relative (i.e. the stake of user x may appear different from the perspectives of users y and z). The model also does not allow for analysis of the 'nothing-at-stake' problem, and does not properly reflect timing differences that exist between PoW and PoS protocols, whereby users who are selected to update state may delay their choice of block to broadcast upon selection. These issues are discussed in more depth in Sect. 2.

As stated in the introduction, Theorem 3 can be seen as a recasting of the CAP Theorem [1,11] for our framework. CAP-type theorems have previously been shown for various PoW frameworks [12,17].

2 The Framework

2.1 The Computational Model

Informal Overview. We use a very simple computational model, designed to be as similar as possible to standard models from distributed computing (e.g. [8]), while also being adapted to deal with the permissionless setting.[3] Processors are specified by state transition diagrams. A *permitter oracle* is introduced as a generalisation of the random oracle functionality in the Bitcoin Backbone paper [10]: It is the permitter oracle's role to grant *permissions* to broadcast messages. The duration of the execution is divided into timeslots. Each processor enters each timeslot t in a given *state* x, which determines the instructions for the processor in that timeslot – those instructions may involve broadcasting messages, as well as sending *requests* to the permitter oracle. The state x' of the processor at the next timeslot is determined by the state x, together with the messages and permissions received at t.

Formal Description. For a list of commonly used variables and terms, see Table 1 in Appendix 2.[4] We consider a (potentially infinite) system of *processors*,

[3] There are a number of papers analysing Bitcoin [10,16] that take the approach of working within the language of the UC framework of Canetti [4]. Our position is that this provides a substantial barrier to entry for researchers in blockchain who do not have a strong background in security, and that the power of the UC framework remains largely unused in the subsequent analysis.

[4] For the appendix, see https://arxiv.org/abs/2101.07095.

some of which may be *faulty*. Each processor is specified by a state transition diagram, for which the number of states may be infinite. At each timeslot t of its operation, a processor p *receives* a pair (M, M^*), where either or both of M and M^* may be empty. Here, M is a finite set of *messages* (i.e. strings) that have previously been *broadcast* by other processors. We refer to M as the *message set* received by p at t, and say that each message $m \in M$ is received by p at t. M^* is a potentially infinite set of pairs (m, t'), where each m is a message and each t' is a timeslot. M^* is referred to as the *permission set* received by p at t. If $(m, t') \in M^*$, then receipt of the permission set M^* means that p is able to broadcast m at step t': Once M^* has been received, we refer to m as being *permitted* for p at t'. To complete the instructions for timeslot t, p then broadcasts a finite set of messages M' that are permitted for p at t, makes a finite *request set* R, and then enters a new state x', where x', M' and R are determined by the present state x and (M, M^*), according to the state transition diagram. The form of the request set R will be described shortly, together with how R determines the permission set received at the next timeslot.

Amongst the states of a processor are a non-empty set of possible *initial states*. The *inputs* to p determine which initial state it starts in. If a variable is specified as an input to p, then we refer to it as *determined* for p, referring to the variable as *undetermined* for p otherwise. If a variable is determined/undetermined for all p, we simply refer to it as determined/undetermined. To define outputs, we consider each processor to have a distinguished set of *output states*, a processor's output being determined by the first output state it enters. Amongst the inputs to p is an *identifier* $\mathtt{U}_p$, which can be thought of as a name for p, and which is unique in the sense that $\mathtt{U}_p \neq \mathtt{U}_{p'}$ when $p \neq p'$. A principal difference between the permissionless setting (as considered here) and the permissioned setting is that, in the permissionless setting, the number of processors is undetermined, and $\mathtt{U}_p$ is undetermined for p' when $p' \neq p$.

We consider a real-time clock, which exists outside the system and measures time in natural number timeslots. We also allow the inputs to p to include messages, which are thought of as having been received by p at timeslot $t = 0$. A *run* of the system is described by specifying the initial states for all processors and by specifying, for each timeslot $t \geq 1$: (1) The messages and permission sets received by each processor at that timeslot, and; (2) The instruction that each processor executes, i.e., what messages it broadcasts, what requests it makes, and the new state it enters.

We require that each message is received by p at most once for each time it is broadcast, i.e. at the end of the run it must be possible to specify an injective function d_p mapping each pair m, t, such that m is received by p at timeslot t, to a triple (p', m, t'), such that $t' < t$, $p' \neq p$ and such that p' broadcast m at t'.

In the *authenticated* setting, we assume the existence of a signature scheme (without PKI), see Appendix 3 for formal details. We let $m_{\mathtt{U}}$ denote the message m signed by $\mathtt{U}$. We consider standard versions (see Appendix 3) of the *synchronous* and *partially synchronous* settings (as in [8]) – the version of the

partially synchronous setting we consider is that in which the determined upper bound Δ on message delay holds after some undetermined stabilisation time.

2.2 The Resource Pool and the Permitter

Informal Motivation. Who should be allowed to create and broadcast new Bitcoin blocks? More broadly, when defining a permissionless protocol, who should be able to broadcast new messages? For a PoW protocol, the selection is made depending on computational power. PoS protocols are defined in the context of specifying how to run a currency, and select identifiers according to their stake in the given currency. More generally, one may consider a scarce resource, and then select identifiers according to their corresponding resource balance.

We consider a framework according to which protocols run relative to a *resource pool*, which specifies a resource balance for each identifier over the duration of the run. The precise way in which the resource pool is used to determine identifier selection is then black boxed through the use of what we call the *permitter oracle*, to which processors can make requests to broadcast, and which will respond depending on their resource balance. To model Bitcoin, for example, we simply allow each identifier (or rather, the processor allocated the identifier) to make a request to broadcast a block at each step of operation. The permitter oracle then gives a positive response with probability depending on their resource balance, which in this case is defined by hashrate. So, this gives a straightforward way to model the process, without the need for a detailed discussion of hash functions and how they are used to instantiate the selection process.

Formal Specification. At each timeslot t, we refer to the set of all messages that have been received or broadcast by p at timeslots $\leq t$ as the *message state* M of p. Each run happens relative to a (determined or undetermined) *resource pool*,[5] which in the general case is a function $\mathcal{R} : \mathcal{U} \times \mathbb{N} \times \mathcal{M} \to \mathbb{R}_{\geq 0}$, where $\mathcal{U}$ is the set of all identifiers and $\mathcal{M}$ is the set of all possible sets of messages (so, $\mathcal{R}$ can be thought of as specifying the resource balance of each identifier at each timeslot, possibly relative to a given message state).[6] For each t and M, we suppose: (a) If $\mathcal{R}(\mathtt{U}, t, M) \neq 0$ then $\mathtt{U} = \mathtt{U}_p$ for some processor p; (b) There are finitely many $\mathtt{U}$ for which $\mathcal{R}(\mathtt{U}, t, M) \neq 0$, and; (c) $\sum_{\mathtt{U}} \mathcal{R}(\mathtt{U}, t, M) > 0$.

After receiving messages and a permission set at timeslot t, suppose p's message state is M_0 and that, for each t', $M^*(t')$ is the set of all messages that are permitted for p at timeslots $\leq t'$. We consider two *settings* – the *timed* and *untimed* settings. The form of each request $r \in R$ made by p at timeslot t depends on the setting, as specified below. While the following definitions might

[5] As described more precisely in Sect. 2.3, whether the resource pool is determined or undetermined will decide whether we are in the *sized* or *unsized* setting.

[6] For a PoW protocol like Bitcoin, the resource balance of each identifier will be their (relevant) computational power at the given timeslot (and hence independent of the message state). For PoS protocols, such as Ouroboros [13] and Algorand [6], however, the resource balance will be determined by 'on-chain' information, i.e. information recorded in the message state M.

initially seem a little abstract, we will shortly give some concrete examples to make things clear.

- **The untimed setting**. Here, each request r made by p must be[7] of the form (M, A), where $M \subseteq M_0 \cup M^*(t)$, and where A is some (possibly empty) extra data. The permitter oracle will respond with a (possibly empty) set M^* of pairs of the form $(m, t+1)$. The value of M^* will be assumed to be a probabilistic function[8] of the determined variables, (M, A), and of $\mathcal{R}(\mathtt{U}_p, t, M)$, subject to the condition that $M^* = \emptyset$ if $\mathcal{R}(\mathtt{U}_p, t, M) = 0$. (If modelling Bitcoin, for example, M might be a set of blocks that have been received by p, or that p is already permitted to broadcast, while A specifies a new block extending the 'longest chain' in M. If the block is valid, then the permitter oracle will give permission to broadcast it with probability depending on the resource balance of p at time t. We will expand on this example below.)
- **The timed setting**. Here, each request r made by p must be of the form (t', M, A), where t' is a timeslot, $M \subseteq M_0 \cup M^*(t')$ and where A is as in the untimed setting. The permitter oracle will respond with a set M^* of pairs of the form (m, t'). M^* will be assumed to be a probabilistic function of the determined variables,[9] (t', M, A), and of $\mathcal{R}(\mathtt{U}_p, t', M)$, subject to the condition that $M^* = \emptyset$ if $\mathcal{R}(\mathtt{U}_p, t', M) = 0$.

If the set of requests made by p at timeslot t is $R = \{r_1, \ldots, r_k\}$, and if the permitter oracle responds with $M_1^*, \ldots, M_k^*$ respectively, then $M^* := \cup_{i=1}^k M_i^*$ is the permission set received by p at its next step of operation.

By a *permissionless protocol* we mean a pair $(\mathtt{S}, \mathtt{O})$, where $\mathtt{S}$ is a state transition diagram to be followed by all non-faulty processors, and where $\mathtt{O}$ is a permitter oracle, i.e. a probabilistic function of the form described above. It should be noted that the roles of the resource pool and the permitter oracle are different, in the following sense: While the resource pool is a variable (meaning that a given protocol will be expected to function with respect to all possible resource pools consistent with the setting), the permitter is part of the protocol description.

How to Understand the Form of Requests (Informal). To help explain these definitions, we consider how to model some simple protocols.

Modelling Bitcoin. To model Bitcoin, we work in the untimed setting, and we define the set of possible messages to be the set of possible *blocks* (in this paper, we use the terms 'block' and 'chain' in an informal sense, for the purpose of giving

[7] To model a perfectly co-ordinated adversary, we will later modify this definition to allow the adversary to make requests of a slightly more general form (see the Appendix 5).

[8] See Appendix 5 for a detailed explanation of what it means to be a 'probabilistic function'.

[9] In the authenticated setting the response of the permitter is now allowed to be a probabilistic function also of $\mathtt{U}_p$. See Appendix 3 for details.

examples). We then allow p to make a single request of the form (M, A) at each timeslot. Here M will be a set of blocks that have been received by p, or that p is already permitted to broadcast. The entry A will be data (without PoW attached) that specifies a block extending the 'longest chain' in M. If A specifies a valid block, then the permitter oracle will give permission to broadcast the block specified by A with probability depending on the resource balance of $\mathtt{U}_p$ at time t (which is p's hashrate, and is independent of M). So, if each timeslot corresponds to a short time interval (one second, say), then the model 'pools' all attempts by p to find a nonce within that time interval into a single request. The higher $\mathtt{U}_p$'s resource balance at a given timeslot, the greater the probability p will be able to mine a block at that timeslot.[10] Note that the resource pool is best modelled as undetermined here, because one does not know in advance how the hashrate attached to each identifier (or even the total hashrate) will vary over time.

Modelling PoS Protocols. The first major difference for a PoS protocol is that the resource balance of each participant now depends on the message state, and may also be a function of time.[11] So, the resource pool is a function $\mathcal{R} : \mathcal{U} \times \mathbb{N} \times \mathcal{M} \to \mathbb{R}_{\geq 0}$. A second difference is that $\mathcal{R}$ is determined, because one knows from the start how the resource balance of each participant depends on the message state as a function of time. Note that advance knowledge of $\mathcal{R}$ *does not* mean that one knows from the start which processors will have large resource balances throughout the run, unless one knows which messages will be broadcast. A third difference is that, with PoS protocols, processors can generally look ahead to determine their permission to broadcast at future timeslots, when their resource balance may be different than it is at present. This means that PoS protocols are best modelled in the timed setting, where processors can make requests corresponding to timeslots t' other than the current timeslot t. To make these ideas concrete, let us consider a simple example.

There are various ways in which 'standard' PoS selection processes can work. Let us restrict ourselves, just for now and for the purposes of this example, to considering blockchain protocols in which the only broadcast messages are blocks, and let us consider a longest chain PoS protocol which works as follows: For each broadcast chain of blocks C and for all timeslots in a set $T(C)$, the protocol being modelled selects precisely *one* identifier who is permitted to produce blocks extending C, with the probability each identifier is chosen being

[10] So, in this simple model, we don't deal with any notion of a 'transaction'. It is clear, though, that the model is sufficient to be able to define what it means for blocks to be *confirmed*, to define notions of *liveness* (roughly, that the set of confirmed blocks grows over time with high probability) and *consistency* (roughly, that with high probability, the set of confirmed blocks is monotonically increasing over time), and to prove liveness and consistency for the Bitcoin protocol in this model (by importing existing proofs, such as that in [10]).

[11] It is standard practice in PoS blockchain protocols to require a participant to have a currency balance that has been recorded in the blockchain for at least a certain minimum amount of time before they can produce new blocks, for example. So, a given participant may not be permitted to extend a given chain of blocks at timeslot t, but may be permitted to extend the same chain at a later timeslot t'.

proportional to their wealth, which is a time dependent function of C. To model a protocol of this form, we work in the timed and authenticated setting. We consider a resource pool which takes any chain C and allocates to each identifier $\mathtt{U}_p$ their wealth according to C as a function of t. Then we can consider a permitter oracle which chooses one identifier $\mathtt{U}_p$ for each chain C and each timeslot t' in $T(C)$, each identifier $\mathtt{U}_p$ being chosen with probability proportional to $\mathcal{R}(\mathtt{U}_p, t', C)$. The owner p of the chosen identifer $\mathtt{U}_p$ corresponding to C and t', is then given permission to broadcast blocks extending C whenever p makes a request $(t', C, \emptyset)$. This isolates a fourth major difference from the PoW case: For the PoS protocol, the request to broadcast and the resulting permission is not block specific, i.e. requests are of the form (t', M, A) for $A = \emptyset$, and the resulting permission is to broadcast *any* from the range of appropriately timestamped and valid blocks extending C. If one were to make requests block specific, then users would be motivated to churn through large numbers of blocks, making the protocol best modelled as partly PoW.

To model a BFT PoS protocol like Algorand, the basic approach will be very similar to that described for the longest chain PoS protocol above, except that certain other messages might be now required in M (such as authenticated votes on blocks) before permission to broadcast is granted, and permission may now be given for the broadcast of messages other than blocks (such as votes on blocks).

2.3 Defining the Timed/Untimed, Sized/Unsized and Single/Multi-permitter Settings

In the previous section we isolated four qualitative differences between PoW and PoS protocols. The first difference is that, for PoW protocols, the resource pool is a function $\mathcal{R} : \mathcal{U} \times \mathbb{N} \to \mathbb{R}_{\geq 0}$, while for PoS protocols, the resource pool is a function $\mathcal{R} : \mathcal{U} \times \mathbb{N} \times \mathcal{M} \to \mathbb{R}_{\geq 0}$. Then there are three differences in the *settings* that are appropriate for modelling PoW and PoS protocols. We make the following formal definitions:

1. **The timed and untimed settings**. This difference between the timed and untimed settings was specified in Sect. 2.2.
2. **The sized and unsized settings**. We call the setting *sized* if the resource pool is determined. By the *total resource balance* we mean the function $\mathcal{T} : \mathbb{N} \times \mathcal{M} \to \mathbb{R}_{>0}$ defined by $\mathcal{T}(t, M) := \sum_{\mathtt{U}} \mathcal{R}(\mathtt{U}, t, M)$. For the unsized setting, $\mathcal{R}$ and $\mathcal{T}$ are undetermined, with the only restrictions being:
   (i) $\mathcal{T}$ only takes values in a determined interval $[\alpha_0, \alpha_1]$, where $\alpha_0 > 0$ (meaning that, although α_0 and α_1 are determined, protocols will be required to function for all possible $\alpha_0 > 0$ and $\alpha_1 > \alpha_0$, and for all undetermined $\mathcal{R}$ consistent with α_0, α_1, subject to (ii) below).[12]

[12] We consider resource pools with range restricted in this way, because it turns out to be an overly strong condition to require a protocol to function without *any* further conditions on the resource pool, beyond the fact that it is a function to $\mathbb{R}_{\geq 0}$. Bitcoin will certainly fail if the total resource balance over all identifiers decreases sufficiently quickly over time, or if it increases too quickly, causing blocks to be produced too quickly compared to Δ.

(ii) There may also be bounds placed on the resource balance of identifiers owned by the adversary.

3. **The multi-permitter and single-permitter settings**. In the *single-permitter* setting, each processor may submit a single request of the form (M, A) or (t, M, A) (depending on whether we are in the timed setting or not) at each timeslot, and it is allowed that $A \neq \emptyset$. In the *multi-permitter* setting, processors can submit any finite number of requests at each timeslot, but they must all satisfy the condition that $A = \emptyset$.[13]

We do not define the general classes of PoW and PoS protocols (although we will be happy to refer to specific protocols as PoW or PoS). Such an approach would be too limited, being overly focussed on the step-by-step operations. In our impossibility results, we assume nothing about the protocol other than basic properties of the resource pool and permitter, as specified by the various settings above. We model PoW protocols in the untimed, unsized, and single permitter settings, with $\mathcal{R} : \mathcal{U} \times \mathbb{N} \to \mathbb{R}_{\geq 0}$. We model PoS protocols in the timed, sized, multi-permitter and authenticated settings, and with $\mathcal{R} : \mathcal{U} \times \mathbb{N} \times \mathcal{M} \to \mathbb{R}_{\geq 0}$. Appendix 4 expands on the reasoning behind these modelling choices. In the following sections, we will see that whether a protocol operates in the sized/unsized, timed/untimed, or multi/single-permitter settings is a key factor in determining the performance guarantees that are possible (such as availability and consistency in different synchronicity settings).

2.4 The Adversary

Appendix 5 gives an expanded version of this subsection and also considers the meaning of probabilisitic statements in detail. In the permissionless setting, we generally consider Byzantine faults, thought of as being carried out with malicious intent by an *adversary*. The adversary controls a fixed set of faulty processors - in formal terms, the difference between faulty and non-faulty processors is that the state transition diagram for faulty processors might not be $\mathtt{S}$, as specified by the protocol. In this paper, we consider a static (i.e. non-mobile) adversary that controls a set of processors that is fixed from the start of the protocol execution. We do this to give the strongest possible form of our impossibility results. We place no bound on the *size* of the set of processors controlled by the adversary. Rather, placing bounds on the power of the adversary in the permissionless setting means limiting their resource balance. For $q \in [0, 1]$, we say the adversary is *q-bounded* if their total resource balance is always at most a q fraction of the total, i.e. for all M, t, $\sum_{p \in P_A} \mathcal{R}(\mathtt{U}_p, t, M) \leq q \cdot \sum_{p \in P} \mathcal{R}(\mathtt{U}_p, t, M)$, where P_A is the set of processors controlled by the adversary.

[13] The names 'single-permitter' and 'multi-permitter' come from the sizes of the resulting permission sets when modelling blockchain protocols. For PoW protocols the the permission set received at a single step will generally be of size at most 1, while this is not generally true for PoS protocols.

2.5 The Permissioned Setting

So that we can compare the permissioned and permissionless settings, it is useful to specify how the permissioned setting is to be defined within our framework. According to our framework, the permissioned setting is exactly the same as the permissionless setting that we have been describing, but with the following differences:

- The finite number n of processors is determined, together with the identifier for each processor.
- All processors are automatically permitted to broadcast all messages, (subject only to the same rules as formally specified in Appendix 2 for the authenticated setting).[14]
- Bounds on the adversary are now placed by limiting the *number* of faulty processors – the adversary is *q-bounded* if at most a fraction q of all processors are faulty.

3 Byzantine Generals in the Synchronous Setting

Recall from Sect. 2.2 that we write $m_{\mathtt{U}}$ to denote the message m signed by $\mathtt{U}$. We consider protocols for solving a version of 'Byzantine Broadcast' (BB). A distinguished identifier $\mathtt{U}^*$, which does not belong to any processor, is thought of as belonging to the *general*. Each processor p begins with a protocol input $\mathtt{in}_p$, which is a set of messages from the general: either $\{0_{\mathtt{U}^*}\}$, $\{1_{\mathtt{U}^*}\}$, or $\{0_{\mathtt{U}^*}, 1_{\mathtt{U}^*}\}$. All non-faulty processors p must give the same output $o_p \in \{0,1\}$. In the case that the general is 'honest', there will exist $z \in \{0,1\}$, such that $\mathtt{in}_p = \{z_{\mathtt{U}^*}\}$ for all p, and in this case we require that $o_p = z$ for all non-faulty processors.

As we have already stipulated, processors also take other inputs beyond their *protocol input* as described in the last paragraph, such as their identifier and Δ – to distinguish these latter inputs from the protocol inputs, we will henceforth refer to them as *parameter inputs*. The protocol inputs and the parameter inputs have different roles, in that the form of the outputs required to 'solve' BB only depend on the protocol inputs, but the protocol will be required to produce correct outputs for all possible parameter inputs.

3.1 The Impossibility of Deterministic Consensus in the Permissionless Setting

In Sect. 2.2, we allowed the permitter oracle $\mathtt{O}$ to be a probabilistic function. In the case that $\mathtt{O}$ is deterministic, i.e. if there is a single output for each input, we will refer to the protocol $(\mathtt{S}, \mathtt{O})$ as deterministic.

[14] It is technically convenient here to allow that processors can still submit requests, but that requests always get the same response (the particular value then being immaterial).

In the following proof, it is convenient to consider an infinite set of processors. As always, though, (see Sect. 2.2) we assume for each t and M, that there are finitely many $\mathtt{U}$ for which $\mathcal{R}(\mathtt{U}, t, M) \neq 0$, and thus only finitely many corresponding processors given permission to broadcast. All that is really required for the proof to go through is that there are an unbounded number of identifiers that can participate *at some timeslot* (such as is true for Bitcoin, or in any context where the adversary can transfer their resource balance to an unbounded number of possible public keys), and that the set of identifiers with non-zero resource balance can change quickly. In particular, this means that the adversary can broadcast using new identifiers at each timeslot. Given this condition, one can then adapt the proof of [7], that a permissioned protocol solving BB for a system with t many faulty processors requires at least $t + 1$ many steps, to show that a deterministic protocol in the permissionless setting cannot always give correct outputs. Adapting the proof, however, is highly non-trivial, and requires establishing certain compactness conditions on the space of runs, which are straightforward in the permissioned setting but require substantial effort to establish in the permissionless setting.

Theorem 1. *Consider the synchronous setting and suppose $q \in (0, 1]$. There is no deterministic permissionless protocol that solves BB for a q-bounded adversary.*

Proof. See Appendix 6 (in the arXiv version).

Theorem 1 limits the kind of solution to BB that is possible in the permissionless setting. In the context of a blockchain protocol (for state machine replication), however, one is (in some sense) carrying out multiple versions of (non-binary) BB in sequence. One approach to circumventing Theorem 1 would be to accept some limited centralisation: One might have a fixed circle of participants carry out each round of BB (involving interactions over multiple timeslots according to a permissioned protocol), only allowing in new participants after the completion of each such round. While this approach clearly does *not* involve a decentralised solution to BB, it might well be considered sufficiently decentralised in the context of state machine replication.

3.2 Probabilistic Consensus

In light of Theorem 1, it becomes interesting to consider permissionless protocols giving *probabilistic* solutions to BB. To this end, from now on, we consider protocols that take an extra parameter input $\varepsilon > 0$, which we call the *security parameter*. Now we require that, for any value of the security parameter input $\varepsilon > 0$, it holds with probability $> 1 - \varepsilon$ that all non-faulty processors give correct outputs.

Appendix 7 explains which questions remain open for probabilistic permissionless protocols in the synchronous setting. For now, in the interests of conserving space, we just briefly mention another negative result:

Theorem 2. *Consider the synchronous and unauthenticated setting. If $q \geq \frac{1}{2}$, then there is no permissionless protocol giving a probabilistic solution to BB for a q-bounded adversary.*

Proof. See Appendix 7.

4 Byzantine Generals with Partially Synchronous Communication

We note first that, in this setting, protocols giving a probabilistic solution to BB will not be possible if the adversary is q-bounded for $q \geq \frac{1}{3}$ – this follows easily by modifying the argument presented in [8], although that proof was given for deterministic protocols in the permissioned setting. For $q < \frac{1}{3}$ and working in the sized setting, there are multiple PoS protocols, such as Algorand,[15] which work successfully when communication is partially synchronous.

The fundamental result with respect to the *unsized* setting with partially synchronous communication is that there is no permissionless protocol giving a probabilistic solution to BB. So, PoW protocols cannot give a probabilistic solution to BB when communication is partially synchronous.[16]

Theorem 3. *There is no permissionless protocol giving a probabilistic solution to BB in the unsized setting with partially synchronous communication.*

Proof. See Appendix 8.

As stated previously, Theorem 3 can be seen as an analog of the CAP Theorem for our framework. While the CAP Theorem asserts that (under the threat of unbounded network partitions), no protocol can be both available and consistent, it *is* possible to describe protocols that give a solution to BB in the partially synchronous setting [8]. The crucial distinction is that such solutions are not required to give outputs until after the undetermined stabilisation time has passed. The key idea behind the proof of Theorem 3 is that, in the unsized and partially synchronous setting, this distinction disappears. Network partitions are now indistinguishable from waning resource pools. In the unsized setting, the requirement to give an output can therefore force participants to give an output before the stabilisation time has passed.

[15] For an exposition of Algorand that explains how to deal with the partially synchronous setting, see [5].

[16] Of course, it is crucial to our analysis here that PoW protocols are being modelled in the unsized setting. It is also interesting to understand why Theorem 3 does not contradict the results of Sect. 7 in [10]. In that paper, they consider the form of partially synchronous setting from [8] in which the delay bound Δ always holds, but is undetermined. In order for the 'common prefix property' to hold in Lemma 34 of [10], the number of blocks k that have to be removed from the longest chain is a function of Δ. When Δ is unknown, the conditions for block confirmation are therefore also unknown. It is for this reason that the Bitcoin protocol cannot be used to give a probabilistic solution to BB in the partially synchronous and unsized setting.

5 Concluding Comments

We close with some questions.

Question 1. What are the results for the timed/untimed, sized/unsized, and the single/multi-permitter settings other than those used to model PoW and PoS protocols? What happens, for example, when communication is partially synchronous and we consider a variant of PoW protocols for which the total resource balance (see Sect. 2.3) is determined?

While we have defined the single-permitter and multi-permitter settings, we didn't analyse the resulting differences in Sects. 3 and 4. In fact, this is the distinction between PoS and PoW protocols which has probably received the most attention in the previous literature (but not within the framework we have presented here) in the form of the 'nothing-at-stake' problem [2]. In the framework outlined in Sect. 2, we did not allow for a mobile adversary (who can make non-faulty processors faulty, perhaps for a temporary period). It seems reasonable to suggest that the difference between these two settings becomes particularly significant in the context of a mobile adversary:

Question 2. What happens in the context of a mobile adversary, and how does this depend on whether we are working in the single-permitter or multi-permitter settings? Is this a significant advantage of PoW protocols?

In the framework we have described here, we have followed much of the classical literature in not limiting the length of messages, or the finite number of messages that can be sent in each timeslot. While the imagined network over which processors communicate does have message delays, it apparently has infinite bandwidth so that these delays are independent of the number and size of messages being sent. While this is an appropriate model for some circumstances, in looking to model such things as sharding protocols [21] it will be necessary to adopt a more realistic model:

Question 3. How best to modify the framework, so as to model limited bandwidth (and protocols such as those for implementing sharding)?

In this paper we have tried to follow a piecemeal approach, in which new complexities are introduced one at a time. This means that there are a number of differences between the forms of analysis that normally take place in the blockchain literature and in distributed computing that we have not yet addressed. One such difference is that it is standard in the blockchain world to consider a setting in which participants may be late joining. A number of papers [12,17] have already carried out an analysis of some of the nuanced considerations to be had here, but there is more to be done:

Question 4. What changes in the context of late joining? In what ways is this different from the partially synchronous setting, and how does this relate to Question 3? How does all of this depend on other aspects of the setting?

References

1. Brewer, E.A.: Towards robust distributed systems. In: PODC, Portland, OR, vol. 7, pp. 343477–343502 (2000)
2. Brown-Cohen, J., Narayanan, A., Psomas, A., Weinberg, S.M.: Formal barriers to longest-chain proof-of-stake protocols. In: Proceedings of the 2019 ACM Conference on Economics and Computation, pp. 459–473 (2019)
3. Buterin, V.: What is Ethereum? Ethereum Official webpage. www.ethdocs.org/en/latest/introduction/what-is-ethereum.html. Accessed 14 2018
4. Canetti, R.: Universally composable security: a new paradigm for cryptographic protocols. In: Proceedings 42nd IEEE Symposium on Foundations of Computer Science, pp. 136–145. IEEE (2001)
5. Chen, J., Gorbunov, S., Micali, S., Vlachos, G.: ALGORAND AGREEMENT: super fast and partition resilient Byzantine agreement. IACR Cryptol. ePrint Arch. **2018**, 377 (2018)
6. Chen, J., Micali, S.: Algorand. arXiv preprint arXiv:1607.01341 (2016)
7. Dolev, D., Strong, H.R.: Authenticated algorithms for Byzantine agreement. SIAM J. Comput. **12**(4), 656–666 (1983)
8. Dwork, C., Lynch, N.A., Stockmeyer, L.: Consensus in the presence of partial synchrony. J. ACM **35**(2), 288–323 (1988)
9. Garay, J., Kiayias, A., Ostrovsky, R.M., Panagiotakos, G., Zikas, V.: Resource-restricted cryptography: revisiting MPC bounds in the proof-of-work era. In: Canteaut, A., Ishai, Y. (eds.) EUROCRYPT 2020. LNCS, vol. 12106, pp. 129–158. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-45724-2_5
10. Garay, J.A., Kiayias, A., Leonardos, N.: The bitcoin backbone protocol: analysis and applications (2018)
11. Gilbert, S., Lynch, N.: Brewer's conjecture and the feasibility of consistent, available, partition-tolerant web services. ACM SIGACT News **33**(2), 51–59 (2002)
12. Guo, Y., Pass, R., Shi, E.: Synchronous, with a chance of partition tolerance. In: Boldyreva, A., Micciancio, D. (eds.) CRYPTO 2019. LNCS, vol. 11692, pp. 499–529. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-26948-7_18
13. Kiayias, A., Russell, A., David, B., Oliynykov, R.: Ouroboros: a provably secure proof-of-stake blockchain protocol. In: Katz, J., Shacham, H. (eds.) CRYPTO 2017. LNCS, vol. 10401, pp. 357–388. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-63688-7_12
14. Lamport, L., Shostak, R., Pease, M.: The Byzantine generals problem. ACM Trans. Program. Lang. Syst. (TOPLAS) **4**(3), 382–401 (1982)
15. Nakamoto, S., et al.: Bitcoin: a peer-to-peer electronic cash system (2008)
16. Pass, R., Seeman, L., shelat, a.: Analysis of the blockchain protocol in asynchronous networks (2016). https://eprint.iacr.org/2016/454.pdf
17. Pass, R., Shi, E.: Rethinking large-scale consensus. In: 2017 IEEE 30th Computer Security Foundations Symposium (CSF), pp. 115–129. IEEE (2017)
18. Pease, M., Shostak, R., Lamport, L.: Reaching agreement in the presence of faults. J. ACM (JACM) **27**(2), 228–234 (1980)

19. Ren, L., Devadas, S.: Proof of space from stacked expanders. In: Hirt, M., Smith, A. (eds.) TCC 2016. LNCS, vol. 9985, pp. 262–285. Springer, Heidelberg (2016). https://doi.org/10.1007/978-3-662-53641-4_11
20. Terner, B.: Permissionless consensus in the resource model. IACR Cryptol. ePrint Arch. **2020**, 355 (2020)
21. Zamani, M., Movahedi, M., Raykova, M.: Rapidchain: scaling blockchain via full sharding. In: Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security, pp. 931–948 (2018)