Resource Pools and the CAP Theorem

ANDREW LEWIS-PYE AND TIM ROUGHGARDEN

Blockchain protocols differ in fundamental ways, including the mechanics of selecting users to produce blocks (e.g., proof-of-work vs. proof-of-stake) and the method to establish consensus (e.g., longest chain rules vs. BFT-inspired protocols). These fundamental differences have hindered "apples-to-apples" comparisons between different categories of blockchain protocols and, in turn, the development of theory to formally discuss their relative merits.

This paper presents a parsimonious abstraction sufficient for capturing and comparing properties of many well-known permissionless blockchain protocols. Our framework blackboxes the precise mechanics of the user selection process, allowing us to isolate the properties of the selection process which are significant for protocol design. We illustrate the framework's utility with two results. First, we prove an analog of the CAP theorem from distributed computing for our framework. This theorem shows that a fundamental dichotomy holds between protocols that are *adaptive*, in the sense that they can function given unpredictable levels of participation, and protocols that have certain *finality* properties. Second, we formalize the idea that proof-of-work (PoW) protocols and non-PoW protocols can be distinguished by the forms of permission that users are given to carry out updates to the state.

1 INTRODUCTION

The task of a permissionless blockchain protocol is to establish consensus for message ordering over a network of users. This job is made difficult by the fact that, being subject to the laws of physics, the underlying communication network must have *latency*, i.e. published messages will necessarily take time to travel over the network of users. As a consequence of this latency, malicious users may purposely cause the order in which messages are first seen to be different for different honest users, and some differences in ordering will anyway be an honest consequence of varying propagation times between different nodes of the network [9].

Network latency is especially problematic when we work at the level of individual transactions, which may be produced at a rate which is high compared to network latency. For this reason it is standard practice to collect transactions together into *blocks*, which can then be produced at a rate which is much lower compared to network latency. Given that we are working in a permissionless setting, the basic question then becomes, "Who should produce the blocks?". This question is answered differently by different protocols. Three running examples, which we shall use for reference in this paper, are:

- Bitcoin;
- Snow White;
- Algorand.

Of course, Bitcoin [14] is the best known proof-of-work (PoW) protocol, and is also a *longest chain* protocol. This means that forks may occur in the blockchain, but that honest miners will build on the longest chain. At a high level, Snow White [1] might be seen as a proof-of-stake (PoS) version of Bitcoin – it is also a longest chain protocol, but now miners are selected with probability proportional to their stake in the currency, rather than their hashing power. We use Algorand [8] as an example of a 'BFT' protocol.[1] This means that users are selected, and asked to carry out a consensus protocol designed for the permissioned setting. So users are not only asked to produce blocks, but also other objects, such as *votes* on blocks. Algorand is also a PoS protocol.

[1]By 'BFT protocols' we shall (informally) mean either: (a) Consensus protocols which are defined in the permissioned setting in order to deal with byzantine faults, or (b) Consensus protocols which work in the permissionless setting by importing protocols of type (a).

Emails. Andrew Lewis-Pye: a.lewis7@lse.ac.uk, andy@lewis-pye.com. Tim Roughgarden: tim.roughgarden@gmail.com

The fundamental differences between these types of protocols has hindered "apples-to-apples" comparisons between them and, in turn, the development of theory to formally discuss their relative merits.

The first and main aim of this paper is to establish a framework for analysing permissionless blockchain protocols that blackboxes the precise mechanics of the user selection process, allowing us to isolate the properties of the selection process that are significant, and to make comparisons between blockchain protocols of different types. Section 2 describes a framework of this kind, according to which protocols run relative to a *resource pool*. This resource pool specifies a balance for each user over the duration of the protocol execution (such as hashrate or stake), which may be used in determining which users are permitted to make publications updating the state.

With this framework in place, we then turn our attention to consider how properties of the resource pool may influence the functionality of the resulting protocol. In Sections 3 and 4, we will be concerned with the distinction between scenarios in which the size of the resource pool is known (e.g. PoS), and scenarios where the size of the resource pool is unknown (e.g. PoW). We refer to these as the *sized* and *unsized* settings, respectively. We will find that the choice of setting is intimately related to a fundamental tradeoff for permissionless blockchain protocols, which can be viewed as an analog of the CAP theorem from distributed computing [11] for our framework: A protocol cannot deliver *finality* for block confirmations while at the same time being *adaptive*, in the sense that it remains live without knowledge of the size of the resource pool.

In Section 5 we will examine a fundamental distinction between PoW and non-PoW protocols, which concerns the forms of permission that users are given to carry out updates to the state. We formalize the idea that, under quite general conditions, PoW protocols are distinguished by their ability to allow the publication of specific blocks (and other objects), rather than granting permission to publish any object from a large class (such as any valid block extending a given position in the blockchain). Historically, this is one of the distinctions between PoW and PoS that has received the most attention in the literature [2].

1.1 Finality and Adaptivity

Our main impossibility result, which appears in Section 4, concerns notions of 'finality', 'adaptivity', 'security', and 'liveness'. Before defining these terms, it is useful to consider how these notions relate to another informal division that is often drawn in the cryptocurrency community and in the literature [2], between 'longest chain' type protocols such as Bitcoin and Snow White on the one hand, and so called 'BFT' protocols such as Algorand [8] and Tendermint [3] on the other.

Roughly speaking, the term 'longest chain' is normally applied to protocols which are derived from Bitcoin, and which work by having miners select a fork of the blockchain to build on, which is defined in terms of some sort of scoring function for chains. The selected chain might be the one with the most PoW attached, or it might be the longest, or it might be defined by an inductive process that counts the number of descendants, as in the GHOST protocol [16]. BFT protocols, on the other hand, work by selecting a subset of users and having them carry out a more traditional consensus protocol which is defined for the permissioned setting. The terms 'BFT' and 'longest chain' are thus descriptive of *where* protocols come from, but don't yet formally define classes of protocols in a way that allows us to analyse the differences between them and prove results contrasting the performance of these classes of protocol in a broad sense.

The informal idea is that there is a trade-off [2]. While BFT protocols potentially offer *finality* (whatever that should mean), this comes with the price that the protocol will stall if participation levels drop. In Algorand, for example, committees of users are selected in rounds, and block confirmation requires a certain proportion of committee members to contribute signatures. If participation levels drop to a point where insufficiently many signatures are being produced for each block, then the process of block confirmation will come to a standstill. Longest chain protocols such as Bitcoin, on the other hand, do not deliver finality but are *adaptive*, in the sense that they naturally adjust and remain live in the face of fluctuating levels of participation.

Intuitively, 'finality' means that blocks of transactions will not be revoked once committed to the blockchain with a suitable level of confirmation. Bitcoin and most other longest chain-type protocols offer 'probabilistic finality' assuming the ongoing participation of a large fraction of honest users: Once a block is suitably confirmed, it is theoretically possible but highly unlikely that it does not belong to the selected chain at some later point in time. In Algorand and most BFT-type protocols, there is essentially zero chance that confirmed blocks will be rolled back, even in the event that honest users cease to participate after the block has been confirmed, e.g. due to an extended period of network failure. Section 3 formalizes this distinction in terms of security (i.e., probabilistic finality) in a *partially synchronous setting*. (The distinction between synchronous and partially synchronous settings is standard when working with permissioned protocols [10].)

We will define the *unsized* setting, so as to formalise contexts in which the total size of the resource pool is information which is not available to the protocol. With this definition in place, and once we have formalised the notion of 'liveness' – roughly, liveness is the property that with high probability the set of confirmed blocks will grow over time – we will then be able to define adaptive protocols as those which are live in the unsized setting. Adaptive protocols are thus those which are like Bitcoin, in the sense that they are live even when the total size of the resource pool is not information which is available to the protocol.

1.2 The tradeoff between adaptivity and finality.

With these definitions in place, we will then be able to formally prove Theorem 4.1 below, which can be seen as an analog of the CAP Theorem [11] from distributed computing for our framework. Roughly speaking, 'security' in our framework corresponds to 'atomic consistency' in the framework in which the CAP Theorem is proved in [11], and 'liveness' corresponds to 'availability'. These correspondences are not exact, however—while availability requires a response even during extended periods of asynchrony, our definition of liveness explicitly rules out the requirement that new confirmed blocks should be produced under such conditions. The key observation in the proof of Theorem 4.1 is that, in the unsized setting, extended periods of asynchrony cannot be distinguished from a waning resource pool. Liveness therefore forces the production of new confirmed blocks during appropriately chosen periods of asynchrony. Liveness and security are thus incompatible in the partially synchronous and unsized setting, while the same is not true in the partially synchronous and sized setting.

Theorem 4.1 (Impossibility Result). *No protocol is both adaptive and has finality.*

This result establishes a simple dichotomy for permissionless blockchain protocols. A protocol can be adaptive or it can have finality, but not both. It also draws a clean and formal line between longest chain protocols such as Bitcoin and Ethereum [4], or PoS implementations such as Snow White [1] on the one hand, and BFT protocols such as Algorand, Casper FFG and PoS implementations of Tendermint or Hotstuff [17] on the other. While the former group are all adaptive, the latter group all have finality.

Another interesting conclusion that can be drawn from Theorem 4.1 concerns PoW protocols. PoS protocols are generally best modeled using the sized setting, while PoW protocols are generally best modeled using the unsized setting—the total stake is typically available to a protocol from the beginning of its execution, while the amount of computational power used to provide PoW can vary over time in an unpredictable way. To the extent that PoW protocols must operate in an unsized setting (and guarantee liveness), Theorem 4.1 implies that they cannot have finality.[2]

1.3 Related Work

The novel feature of Bitcoin that distinguishes it from previous consensus protocols is that it is *permissionless*, i.e. it establishes consensus between a set of users that anybody can join, with as many identities as they choose in any given role. This paper can be seen as a step towards developing the same sort of formal framework for the analysis of permissionless protocols, as has been extensively developed for permissioned protocols over a number of decades [13]. The study of byzantine fault tolerant (BFT) consensus protocols in the permissioned setting dates back at least to 1980 [12, 15]. Among those BFT protocols of interest to us here, we can distinguish two forms:

(1) The oldest relevant form of BFT protocol is aimed at solving the 'Byzantine Generals Problem' [12, 15]. The task here is to reach consensus on a single yes/no decision. In applying these methods to the blockchain setting, one approach, as employed by Algorand, is to run such a BFT protocol for each in a sequence of proposed blocks, until consensus is reached for each block as to whether it should be included in the blockchain. A drawback of this approach is that positive or negative consensus has to be reached for one block at a time. A large number of rounds of communication may be required before agreement is reached, giving a corresponding negative impact on confirmation times.

(2) Perhaps more appropriate for the blockchain setting, since they are designed to achieve essentially the same task as permissionless blockchain protocols but in the permissioned setting, are BFT protocols designed for the purpose of state machine replication (SMR) [6, 7]. The task of such protocols is for a set of distributed users to agree on an order of execution for an ever growing list of client-initiated service commands – replace 'client-initiated service commands' with 'transactions', and this is precisely the aim of permissionless blockchain protocols. The advantage of this approach is that it allows for considerably simpler protocols, which might only require two or three rounds of communication per block.

The CAP theorem is one of the most celebrated theorems in the distributed computing literature. The theorem proved by Gilbert and Lynch [11] is a formal version of a conjecture due to Brewer, which is made in the context of distributed web services. The theorem establishes an impossibility result: It is impossible for such a distributed service to simultaneously achieve the three desirable properties of *consistency, availability*, and *partition tolerance*. For formal definitions of these terms we refer the reader to [11] and [13]. The relationship between the CAP Theorem and Theorem 4.1 was briefly discussed in Section 1.2, and we shall expand on this discussion in Section 4.

2 THE FRAMEWORK

2.1 Predetermined and undetermined variables

Our aim here is to establish a framework for analysing permissionless blockchain protocols that blackboxes the precise mechanics of the user selection process. This will allow us to prove impossibility results, and to isolate the properties of the selection process which are significant, in the sense that they impact the way in which the protocol must be designed, or influence properties of the resulting protocol (such as security in a range of settings).

In order to define properties such as liveness and security later on, it will be convenient to consider protocols that are specified relative to a finite set of initially defined *parameters*. For Algorand to run securely, for example, one must first decide how long the protocol is to run for,

[2]The continual adjustment of the difficulty parameter in Bitcoin can be viewed as an attempt to maintain an approximation of the sized setting in a fundamentally unsized setting. See Section 6.2 for further discussion.

and then choose committee sizes accordingly. The duration of the execution is therefore required as a parameter of the protocol. Variables that are specified before the execution of the protocol as parameters, or which take the same value for all executions of the protocol, are referred to as *predetermined*. Variables (such as the number of users) that are not predetermined, will be referred to as *undetermined*.

2.2 The users

We consider settings in which protocols are executed by an undetermined set of pseudonymous users, this set being of undetermined size. Each user is given access to a signature scheme, and controls a set of public keys by which they will be known to other users. We use the variable U to range over users, while $\mathtt{U}$ will be used to range over public keys – so each user may have many public keys. Amongst all users, there is one who is distinguished as the *adversary* and who controls an undetermined set of public keys.

We will suppose that each user is a deterministic computing device, which has amongst the actions it can perform calls to certain oracles, as well as certain external functionalities such as the ability to *publish* messages. The protocol will be a program which is run by every user, other than the adversary. The adversary can follow any program of their choosing.

While we might think of the set of users as forming a network over which messages can propagate, in order to keep things as simple as possible, we shall not make the network explicit in our framework. Users simply have the ability to *publish* messages. Once a message is published by a public key belonging to a given user, it may subsequently be *delivered* to other users at different stages of the execution.

2.3 Network failures

We suppose that protocols are specified to run for a predetermined sequence of timeslots, each timeslot being of predetermined length. The appropriate length of these timeslots depends on the protocol to be modeled. For many PoS protocols, an appropriate length is slightly more than the network latency, i.e. the time it takes a block to propagate the network. (Thus, each user might carry out many instructions during a single timeslot.) We will see that PoW protocols might be better modeled using very short timeslots. In any case, the sequence of timeslots is called the *duration* $\mathcal{D}$. At the beginning of each timeslot in the duration, published messages may be delivered to various users. The *published state* relative to a given user is the set of all published messages which have been delivered to them. The published state for a given user is therefore monotonically increasing over time. In order to be published, a message must be *valid*, meaning that it must have a certain structure and that certain other conditions, expanded on below, are also satisfied.

For example, if modeling Bitcoin or Snow White, a user's published state will be the set of (valid) blocks that have been delivered to them. Thus the published state will not, in general, be a single chain of blocks. For Algorand, a user's state will be all those messages which have been delivered to them, which are either valid blocks, or else the signed messages of committee members exchanged while reaching consensus on blocks.

It is standard in the distributed computing literature to consider a variety of *synchronous, partially synchronous*, or *asynchronous* settings, in which users may or may not have clocks which are almost synchronised, or run at varying speeds, and where message delivery might be reliable or subject to various forms of failure. For the sake of simplicity, we will suppose here that users' clocks are synchronised – while this might seem like a strong assumption, this only strengthens our impossibility result. We will, though, allow for periods of network failure, during which the adversary is able to control message delivery. In order to formalise this, we will suppose that the duration is divided into *synchronous* and *asynchronous* intervals (meaning that each timeslot is either synchronous or asynchronous). We will suppose that, during synchronous intervals, message delivery time is probabilistically distributed for each pair of users. During asynchronous intervals, we suppose that the adversary is able to interfere with message delivery as they choose, i.e. the adversary can leave messages to be delivered in a probabilistic fashion as normal, can cause undelivered messages to be delivered early, or can stop messages being delivered at all for the duration of the asynchronous interval. It will be convenient, however, to suppose that messages are always delivered instantaneously to the user who publishes them. We then distinguish two *synchronicity settings*. In the *synchronous* setting it is assumed that there are no asynchronous intervals during the duration, while in the *partially synchronous* setting there may be undetermined asynchronous intervals.

2.4 The structure of the blockchain

Amongst all published messages, there is a distinguished set referred to as *blocks*, and one block which is referred to as the *genesis block*. Unless it is the genesis block, each block B has a unique *parent* block $\texttt{Par}(B)$, which must be uniquely specified within the block message. Each block is produced by a single user, $\texttt{Miner}(B)$, but may contain other published messages which have been produced by other users. No block can be published by $\texttt{U} := \texttt{Miner}(B)$ at a point strictly prior to that at which its parent has been delivered to $\texttt{U}$. $\texttt{Par}(B)$ is defined to be an *ancestor* of B, and all of the ancestors of $\texttt{Par}(B)$ are also defined to be ancestors of B. If B is not the genesis block, then it must have the genesis block as an ancestor. At any point during the duration, the set of published blocks thus forms a tree structure. If P is a published state, then we shall say that it is *downward closed* if it contains the parents of all blocks in P. By a *leaf* of P, we shall mean a block in P which is not a parent of any block in P. If P is downward closed and contains a single leaf, then we shall say that P is a *chain*.

2.5 The resource pool

Protocols are run relative to a (predetermined or undetermined) *resource pool*, which in the general case is a function $\mathcal{R} : \mathcal{U} \times \mathcal{D} \times \mathcal{P} \rightarrow \mathbb{R}_{\geq 0}$, where $\mathcal{U}$ is the set of public keys, $\mathcal{D}$ is the duration and $\mathcal{P}$ is the set of all possible published states. So $\mathcal{R}$ can be thought of as specifying the resource balance of each user at each timeslot in the duration, possibly relative to a given published state. For a PoW protocol like Bitcoin, the resource balance of each public key will be their (relevant) computational power at the given timeslot. For PoS protocols, such as Snow White and Algorand, however, the resource balance will be fully determined by 'on-chain' information, i.e. information recorded in the published state P. Generally, a chain of blocks $C \subseteq P$ will first be selected. So C might be the longest chain, or the longest chain of blocks that have been approved by committee members. Then $\mathcal{R}(\texttt{U}, t, P)$ will be some function of $\texttt{U}$'s stake as recorded by the blocks in C.[3]

By the *total resource balance* $\mathcal{T}$, we mean the sum of the resource balances of all public keys; this is the function $\mathcal{T} : \mathcal{D} \times \mathcal{P} \rightarrow \mathbb{R}_{\geq 0}$ defined by $\mathcal{T}(t, P) = \sum_{\texttt{U}} \mathcal{R}(\texttt{U}, t, P)$. It should be noted that the resource pool is a variable, meaning that a given protocol may be expected to be live and secure with respect to a range of resource pools.

2.6 The sized and unsized settings

Just as we considered two synchronicity settings earlier, we also consider two *resource settings*. The basic idea is that in the *sized* setting, the total resource balance is information which is available to

[3]The details here will depend on the specific protocol. It's standard to insist that a user has had stake in the currency recorded for a certain number of timeslots before they are allowed to produce blocks, for example. So $\texttt{U}$'s resource balance might be their stake according to C or some initial segment of C, or else 0 if $\texttt{U}$ has not been recorded as having non-zero stake for sufficient time.

the protocol (and the permitter, as described in Section 2.7), while in the *unsized* setting it is not. The precise details are as follows.

The unsized setting. For the unsized setting, $\mathcal{R}$ (and hence $\mathcal{T}$) is undetermined, with the only restrictions being:

(1) $\mathcal{R}$ will be a function from $\mathcal{U} \times \mathcal{D} \times \mathcal{P}$ to $\mathbb{R}_{\geq 0}$ satisfying the requirement that, at all timeslots in the duration, the total resource balance belongs to a fixed interval $[I_0, I_1]$, where $I_0 > 0$ is sufficiently small and $I_1 > I_0$ is sufficiently large.[4]
(2) There may also be bounds placed on the resource balance of the adversary.

We shall refer to the set of all resource pools satisfying these restrictions as the *possible resource pools*, and in Section 3 we shall define a protocol to be live if is live for all possible resource pools.

The sized setting. For the sized setting, the total resource balance $\mathcal{T}$ is a predetermined function $\mathcal{T} : \mathcal{D} \times \mathcal{P} \rightarrow \mathbb{R}_{\geq 0}$.

The basic idea is that PoS protocols will generally be best modeled using the sized setting, while PoW protocols are best modeled using the unsized setting, since one does not know the total resource balance (e.g., total hashrate in each timeslot) in advance. There are some nuanced considerations, however. With a PoS protocol, for example, one might not be able to predict accurately what percentage of the stake will actually come online and publish as requested by the protocol. So there may be situations in which it is appropriate to define the resource balance in terms of the online or contributing stake, and where it should be recognised that only partial information will be available concerning the total resource balance. Equally, there may be contexts in which good bounds can be given on the total resource balance over the duration for the PoW case. The example of Bitcoin will be discussed further in Section 6.

2.7 The permitter oracle

In order to specify how the resource pool is to be used, we shall make use of the notion of a *permitter oracle*. This is the most critical part of the model, and is the part that blackboxes user selection, since it is the permitter oracle that grants permission to publish valid messages. The permitter oracle need not be implemented explicitly in the blockchain being modeled, and is a mathematical abstraction that allows for the discussion and comparison of blockchains of very different types. It is designed to be as simple as possible, subject to this goal.

As described in Section 2.2, we consider each user to be a computing device with access to certain external oracles and functionalities. At any given timeslot $t \in \mathcal{D}$, a user's *state* is entirely specified by the set of public keys they control, the protocol parameters, their published state and the set of permissions they have been given by the permitter. The *protocol* P is then a set of deterministic and efficiently computable instructions, which specifies precisely what actions a user should carry out at each timeslot, as a function of the timeslot and their state at that timeslot. The instructions of the protocol are therefore a function of the timeslot, the keys controlled by the user, the protocol parameters, their published state, and the set of permissions they have been given by the permitter.

One of the external functionalities each user has is the ability to publish valid messages. Amongst the conditions required for validity is that the public key responsible for the publication has been given *permission* by the permitter oracle O, which is an oracle to which users have access. We thus

[4]We consider resource pools with range restricted to the fixed interval $[I_0, I_1]$ because it turns out to be an overly strong condition to require a protocol to be live without *any* further conditions on the total resource balance, beyond the fact that it is a function to $\mathbb{R}_{\geq 0}$. We wish to be able to talk about Bitcoin as live in the unsized setting, for example, but liveness will certainly fail if $\mathcal{R}$ is the constant 0 function, or if the total resource balance decreases sufficiently quickly over time.

suppose that users can make 'requests' to the permitter, of the form (U, P, t', A), where U is a public key under their control, P is a possible downward closed published state, t' is a timeslot, and where A is some (possibly empty) extra data. Given a request of this form, the permitter may then respond by giving them permission to publish certain messages. The response of the permitter to a request (U, P, t', A) will be assumed to be a probabilistic function of the protocol parameters, the actual timeslot t, the previous requests made by U, the tuple (U, P, t', A), and of the user's resource level $\mathcal{R}(\mathsf{U}, t', P)$.[5]

The conditions we have described above non-trivially restrict what the permitter can do. For example, consider the unsized setting, and suppose that the total resource pool (e.g., total hashrate) cannot be deduced from the protocol parameters, t, previous requests made by U, the tuple (U, P, t', A), and the user's resource level $\mathcal{R}(\mathsf{U}, t', P)$. In this case, the framework requires that the response of the permitter be independent of the total resource pool. (Whereas in the sized setting, if the total resource pool can be deduced from the published state P, the permitter is not so constrained.)

As we shall discuss later, the form of the permission given by the permitter might be permission to publish a specific message (such as the data A proposed by the user in their request), or it might be permission to publish any number of messages satisfying certain criteria (such as any block that extends the published state at a given location). In what follows we shall consider various settings, depending on what assumptions can be made about the relationship between the permitter and the resource pool.

It should be noted that the roles of the resource pool and the permitter are different in the sense that, while the resource pool is a variable (meaning that a given protocol may be expected to be live and secure with respect to a range of resource pools), protocols are generally only required to run relative to a specific permitter oracle.

2.8 Modeling simple PoW and PoS protocols

For concreteness, we next consider how some simple PoS and PoW protocols can be modeled using our framework. We remind the reader that our goal is not to literally model the step-by-step operation of these protocols, but rather to replicate the essential properties of their user selection mechanisms with a suitable choice of a permitter oracle.

First, consider a PoW protocol like Bitcoin. To keep things simple, we'll initially ignore Bitcoin's adjustable 'difficulty parameter' (i.e., how hard the PoW is to produce); we'll return to this point in Section 6. To model a simple PoW protocol of this form, we can consider very short timeslots (say 1 second each, or even shorter). The resource level (i.e., hashrate) of a user in a given timeslot is independent of the published state, so we can restrict attention to resource pools $\mathcal{R} : \mathcal{U} \times \mathcal{D} \rightarrow \mathbb{R}_{\geq 0}$. We interpret a user request (U, P, t', A) in a timeslot t as all of U's efforts during timeslot t to extend the published state P.[6] (If a user submits more than one request during a timeslot, the permitter ignores all but the first.) For example, we can interpret A as a choice and ordering of transactions within a proposed block, along with a choice of predecessor, with the understanding that the user will try as many different nonces as possible during the timeslot. The permitter then gives U permission to publish with probability proportional to $\mathcal{R}(\mathsf{U}, t)$ (so long as A can be legally added to P).[7] A notable feature of this permitter is that permission is granted for the publication of specific

[5] Another way to interpret these conditions is that the response to a request (or the probability distribution on that response) should be fully determined by information known to the user making the request – in practice, users should be able to check for themselves whether or not they have permission to publish.

[6] The parameter t' is ignored by the permitter, or equivalently is automatically interpreted as the current timeslot t; the parameter t' is relevant only for PoS protocols.

[7] For example, with probability $\mathcal{R}(\mathsf{U}, t)/M$, where M is large constant that depends on the assumed maximum hashrate I_1 and the timeslot length.

messages (i.e., a specific choice of A), rather than for a collection of messages meeting certain criteria.

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 protocols in which the only published messages are blocks, and let us consider a longest chain PoS protocol which works as follows: For each published chain C and for all timeslots in a set $T(C)$, the protocol being modeled selects precisely *one* public key who is permitted to produce blocks extending C (i.e. blocks whose parent is the unique leaf of C), with the probability each public key is chosen being proportional to their wealth as recorded in C.[8] In order to model a protocol of this form, we can consider a (timeslot-independent) resource pool $\mathcal{R} : \mathcal{U} \times \mathcal{P} \rightarrow \mathbb{R}_{\geq 0}$, which takes the longest chain C from P, and allocates to each public key $\mathtt{U}$ their wealth according to C.[9] Then we can consider a permitter which chooses one public key $\mathtt{U}$ for each chain C and each timeslot t' in $T(C)$, each public key $\mathtt{U}$ being chosen with probability $\mathcal{R}(\mathtt{U}, C)/\mathcal{T}(C)$. (This is well defined because the total resource pool $\mathcal{T}$ is known to the protocol.) That chosen public key $\mathtt{U}$ corresponding to C and t', is then given permission to publish blocks extending C whenever $\mathtt{U}$ makes a request $(\mathtt{U}, P, t', \emptyset)$ for which C is the longest chain in P. A notable feature of this permitter is that the permission it gives is for the publication of *sets* of messages satisfying certain criteria, i.e. when the permitter gives permission it is for any (otherwise valid) block extending a given chain C.

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

3 ADAPTIVITY AND FINALITY

3.1 The extended protocol and the meaning of probabilistic statements

In order to define what it means for a protocol to be secure or live, we first need a *notion of confirmation* for blocks, which is a function $\mathtt{C}$ mapping any published state to a chain which is a subset of that published state, in a manner which depends on an initially defined parameter called the *security parameter* $\varepsilon \geq 0$. The intuition behind ε is that it should upper bound the probability of false confirmation. Given any published state, $\mathtt{C}$ returns the set of confirmed blocks.

In Sections 2.2 and 2.7, we stipulated that the protocol is a set of deterministic and efficiently computable instructions, which specifies precisely what actions a user should carry out at each timeslot. In general, however, a protocol might only be considered to run relative to a single specific permitter oracle $\mathtt{O}$, and a specific notion of confirmation $\mathtt{C}$. We will refer to the triple $(\mathtt{P}, \mathtt{O}, \mathtt{C})$ as the *extended protocol*. Each execution of the extended protocol is then entirely determined by:

(1) The parameters;
(2) The set of users and their public keys;
(3) An index specifying the program executed by the adversary;
(4) The resource pool (which may or may not be undetermined);
(5) The set of asynchronous timeslots;

[8] Note that being permitted to publish a block is not the same as being instructed by the protocol to publish a block, and does not determine how other users will treat the block – there are many contexts in which users might be able to produce valid blocks for which publication is not instructed by the protocol. A user may also be permitted to produce two valid blocks whose publication constitutes an overt deviation from the protocol, and which might be punished.

[9] Note that in many PoS protocols the relevant balance is actually $\mathtt{U}$'s wealth according to some proper initial segment of C, and that in modeling such protocols one should adjust $\mathcal{R}$ accordingly. As mentioned earlier, it is also standard to insist that $\mathtt{U}$ has been recorded as a public key with non-zero stake for a minimum number of timeslots.

(6) Certain events on which probability distributions are established, including permitter responses and message delivery times.

Generally, when we discuss an extended protocol, we shall do so within the context of a *setting*, which constrains the set of possible choices for (1)-(6) above. The setting might specify the probability distribution on delivery times, for example, and might restrict the set of resource pools to those in which the adversary is given a limited resource balance. When we make a probabilistic statement to the effect that a certain condition holds with at most/least a certain probability, this means that the probabilisitic bound holds for all possible values of (1)-(5) above that are not made explicit in the statement, and which are consistent with the setting.

3.2 Liveness and adaptivity

In order to define liveness for a protocol with a notion of confirmation C, let $|\mathsf{C}(P)|$ denote the number of blocks in $\mathsf{C}(P)$ for any published state P. For a given U, and timeslots $t_1 < t_2$, let P_i be U's published state at t_i. Let us say that $[t_1, t_2]$ is a *growth interval* for U, if $|\mathsf{C}(P_2)| > |\mathsf{C}(P_1)|$.

Definition 3.1. *A protocol is* ***live*** *if, for every choice of security parameter $\varepsilon > 0$, there exists ℓ_ε such that the following holds with probability at least $1 - \varepsilon$ for any timeslots $t_1 < t_2 \in \mathcal{D}$, and for any honest U: If $t_2 - t_2 \geq \ell_\varepsilon$ and $[t_1, t_2]$ is entirely synchronous, then $[t_1, t_2]$ is a growth interval for U.*

So, roughly speaking, a protocol is live if the number of confirmed blocks can be relied on to grow over time during synchronous intervals of sufficient length. In order to properly understand Definition 3.1, we refer the reader to the conventions concerning the meaning of probabilistic assertions that were described in Section 3.1. Generally, assertions of liveness and security will be made within the confines of a particular setting, which might restrict the probability distribution on message delivery times, or limit the resource balance of the adversary (but is otherwise worst-case subject to these constraints).

In order to digest Definition 3.1, it is useful to understand why it should be satisfied by a protocol like Bitcoin. Suppose we model Bitcoin in the unsized and synchronous setting. According to Section 2.6, this means that we assume the existence of a fixed interval $[I_0, I_1]$ such that $I_0 > 0$, $I_1 > I_0$, and such that the total resource balance always takes values in $[I_0, I_1]$. Let us suppose that we model Bitcoin and the permitter as discussed in Section 2.8 – again, for the sake of simplicity, we'll forget about the fact that Bitcoin makes adjustments to the 'difficulty parameter'. Suppose also that, as part of the setting, we assume:

(A) The adversary only ever controls a suitably small proportion of the total resource balance, and;

(B) The probability distribution on the length of time for message delivery is such that the probability of delivery failure tends to 0 as the time after publication tends to ∞.

In order for a block B to be confirmed, C requires that it should belong to the longest chain in P and be followed by x many blocks, where the value of x is a function of the security parameter ε (and possibly the assumed restriction on the adversary's resource balance). Suppose that, at a given timeslot t, C is the longest chain seen by U. Since we assume that the total resource balance always belongs to $[I_0, I_1]$, this allows us to find ℓ^*_ε (independent of t and C) such that the following holds with probability $> 1 - \varepsilon/2$: Some honest user U' is permitted to publish a new block at a timeslot before $t + \ell^*_\varepsilon$, but after all blocks in C have been delivered to them. In order to specify the value ℓ_ϵ whose existence is required by Defintion 3.1, we can then define $\ell_\varepsilon > \ell^*_\varepsilon$ such that, with probability $> 1 - \varepsilon/2$, the block published by U' will be delivered to U by timeslot $t + \ell_\varepsilon$. It then holds with probability $> 1 - \varepsilon$ that the longest chain (and hence the number of confirmed transactions) seen by U at $t + \ell_\varepsilon$ is of length greater than $|C|$.

Now that we have defined liveness, we can also define adaptivity:

Definition 3.2. *We define a protocol to be* ***adaptive*** *if it is live in the unsized setting.*

3.3 Security and finality

Roughly speaking, *security* requires that confirmed blocks normally belong to the same chain. Let us say that two distinct blocks are *incompatible* if neither is an ancestor of the other, and *compatible* otherwise. If $B \in \mathtt{C}(P)$ where P is the published state of $\mathtt{U}$ at time t, then we shall say that B is confirmed for $\mathtt{U}$ at t.

Definition 3.3. *A protocol is* ***secure*** *if the following holds for every choice of security parameter $\varepsilon > 0$, for every $\mathtt{U}_1, \mathtt{U}_2$ and for all timeslots t_1, t_2 in the duration: With probability $> 1 - \varepsilon$, all blocks which are confirmed for $\mathtt{U}_1$ at t_1 are compatible with all those which are confirmed for $\mathtt{U}_2$ at t_2.*

Definition 3.4. *A protocol has* ***finality*** *if it is secure in the partially synchronous setting.*

Note that BFT protocols such as Algorand are normally designed to have finality in this sense. For Algorand, the duration and adversary resource bound are initially specified as a parameters, and then the protocol specifies committee sizes and other quantities so that the probability two incompatible blocks will ever be confirmed is less than ε.

4 THE IMPOSSIBILITY OF ADAPTIVITY AND FINALITY

In Section 2.7, we didn't describe any conditions requiring that the behaviour of the permitter *must* be influenced by the resource pool. The only assumption of this kind that we shall make is stated below, and will be applied for both the sized and unsized settings.

> **No balance, no voice**: No $\mathtt{U}$ will be given permission to publish messages in response to a request $(\mathtt{U}, P, t', A)$ for which $\mathcal{R}(\mathtt{U}, t', P) = 0$.

Now that the framework and all required definitions are in place, we can formally prove Theorem 4.1.

Theorem 4.1. *No protocol is both adaptive and has finality.*

As stated previously, this theorem can be seen as an analog of the CAP Theorem [11] from distributed computing for our blockchain protocol analysis framework. Now that we have formally defined adaptivity, finality, security, and liveness, it may be useful to say a little more about the relationship to the CAP Theorem. While the CAP Theorem asserts that (under the threat of unbounded network partitions), no protocol can be both available and consistent, it is possible for BFT protocols such as Algorand to be both live and secure *in the partially synchronous setting*. This is possible because liveness is a fundamentally weaker property than availability: Liveness does not require new confirmed blocks to be produced during extended periods of asynchrony. For example, Algorand is live, even though block production may stop during network partitions. The key idea behind the proof of Theorem 4.1 is that, in the unsized (and partially synchronous) setting, this fundamental difference disappears, with network partitions indistinguishable from waning resource pools. Liveness then forces the existence of growth intervals during network partitions. In the unsized and partially synchronous setting, security and liveness thus become incompatible, just as consistency and availability are incompatible according to the CAP Theorem.

Proof. (of Theorem 4.1) The idea behind the proof can be summed up as follows. We consider executions of the protocol in which there are at least two users, both of which are honest, and who control public keys $\mathtt{U}_0$ and $\mathtt{U}_1$ respectively. Suppose that, in an execution of the protocol in the unsized and partially synchronous setting, $\mathtt{U}_0$ and $\mathtt{U}_1$ both have the same constant and non-zero resource balance, and that all other users have resource balance zero throughout the duration. According to the assumption 'no balance, no vote', this means that U_0 and U_1 will be the only public keys which are able to publish messages. For as long as the adversary is able to prevent messages published by each U_i from being delivered to U_{1-i} ($i \in \{0,1\}$), the execution will be indistinguishable, as far as U_i is concerned, from one in which only U_i has the same constant and non-zero resource balance. The fact that the protocol is live means that, with high probability, U_0 and U_1 will see confirmed blocks within a bounded period of time. The confirmed blocks for U_0 will be incompatible with those for U_1, so long as these confirmed blocks appear before any point at which a message published by U_i has been delivered to U_{1-i} for some $i \in \{0,1\}$. This contradicts security for the protocol in the partially synchronous setting.

To describe the argument in more detail, let U_0 and U_1 be public keys controlled by different honest users. For a duration $\mathcal{D}$ which is sufficiently long, we consider three different resource pools:

$\mathcal{R}_0$: We let $\mathcal{R}_0$ assign the constant value $I > 0$ to both U_0 and U_1 over the entire duration, while all other users are assigned the constant value 0.

$\mathcal{R}_1$: We let $\mathcal{R}_1$ assign the constant value I to U_0 over the entire duration, while all other users are assigned the constant value 0.

$\mathcal{R}_2$: We let $\mathcal{R}_2$ assign the constant value I to U_1 over the entire duration, while all other users are assigned the constant value 0.

We consider three different executions of the protocol with the same parameters, for the unsized setting in which the resource pool is an undetermined variable:

Ex_0 : Here $\mathcal{R} := \mathcal{R}_0$. All timeslots are asynchronous and the adversary prevents the delivery of messages published by U_i to the user controlling U_{1-i}, for $i \in \{0,1\}$.

Ex_1 : Here $\mathcal{R} := \mathcal{R}_1$, and we work in the synchronous setting (or in the partially synchronous setting, but without interference by the adversary).

Ex_2 : Here $\mathcal{R} := \mathcal{R}_2$, and we work in the synchronous setting.

According to the assumption of 'no balance, no voice', it follows that only U_0 and U_1 will be able to publish messages in any of these three executions. Our framework stipulates that the instructions of the protocol for a given user at a given timeslot must be a deterministic function of the protocol parameters, the timeslot, the keys controlled by the user, their published state and the set of permissions they have been given by the permitter (see Section 2.7). It also stipulates that the response of the permitter to a request (U, P, t', A) is a probabilistic function of the protocol parameters, the actual timeslot t, previous requests made by U, the request (U, P, t', A), and the user's resource level $\mathcal{R}(\mathsf{U}, t', P)$. It therefore follows by induction on timeslots that, because the resource pool is undetermined:

(†) For each $i \in \{0,1\}$, and for all timeslots in Ex_0, the probability distribution on the state of the user controlling U_i is identical to the corresponding distribution at the same timeslot in Ex_{1+i}.

If the protocol is adaptive, then it follows from Definition 3.1 that we can find a timeslot t_0 satisfying the following condition: In both Ex_{1+i} ($i \in \{0,1\}$), it holds with probability $> 3/4$ that there is at least one block which is confirmed for U_i at t_0. By (†) it then holds for Ex_0, and for each $i \in \{0,1\}$, that with probability $> 3/4$ there is at least one block which is confirmed for U_i at t_0. We stipulated in Section 2.4 that no block B can be published by $\mathsf{U} := \mathtt{Miner}(B)$ at a point strictly prior to that at which its parent has been delivered to U. It follows that in Ex_0 all blocks which are confirmed for U_i must be incompatible with all blocks which are confirmed for U_{1-i}. The definition of security therefore fails to hold for timeslot t_0, and with respect to the security parameter 1/2. □

5 PROOF-OF-STAKE REQUIRES MULTI-PERMITTERS

One major difference between typical PoW and PoS longest-chain protocols (e.g., Bitcoin vs. Snow White) is the order of operations between a user choosing a proposed block to publish and learning whether or not it has permission to publish. In the dominant PoW protocols, the proposed block is chosen first, and only then is permission granted or denied; in typical longest-chain PoS protocols, permission (to publish in a given timeslot at a given location) is granted before the specific block to publish is chosen. Is this difference an artefact of the protocols developed thus far, or is it a more fundamental distinction between PoW and non-PoW protocols? We next use our framework to reason about this question.

Already from the simple examples in Section 2.8, one can see that the standard PoW and PoS protocols are best modeled by permitters and resource pools with quite different properties. The permitter which we described in modeling the PoS case, for example, was able to ensure that a single user would be given permission to extend a particular chain at a particular timeslot, simply because it has access to the total resource balance recorded by a given chain. As alluded to above, another notable difference is that the permitter we described for the PoW case gave permission for the publication of specific messages, rather than for *sets* of messages satisfying certain criteria (and of size larger than 1). We shall refer to permitters of this type as *single-permitters*, rather than *multi-permitters*.

A key factor in determining whether multi-permitting is inherent to non-PoW protocols is the number of possible blocks extending a given chain C—by the 'possible' extensions of a chain C, we mean those blocks B satisfying all conditions required for validity other than being permitted by the permitter oracle. If the number of possible extensions is large, while the probability that the permitter gives permission for each is small, then a user may be able to increase their probability of gaining permission to publish a block by churning through as many requests as possible. This means that the probability of success comes to depend on computational power, rendering the protocol (at least partially) PoW.

In order to see this more precisely, we need a precise way to talk about the computational power of a user. So, for the purposes of this discussion, let us say that the computational power of a user is the number of requests they are capable of making to the permitter in each timeslot. We'll denote the computational power of U by X_{U}. In order to restrict to realistic scenarios, we'll suppose that there is some fixed upper bound $X_{\max}$, for which we always have $X_{\mathsf{U}} \leq X_{\max}$. Suppose that, at a given timeslot t, C is the longest chain, and, for the sake of simplicity, suppose that C has been seen by all users. Suppose further, that the following conditions are satisfied:

- ($\dagger_1$) The permitter O is a single-permitter. More specifically, let Λ be the set of requests of the form (U, C, t, B), such that B is a possible extension of C. There exists some $\lambda > 0$, such that O will respond to each distinct request in Λ made during timeslot t, by giving permission to publish the specific block B with independent probability $\lambda \cdot \mathcal{R}(\mathsf{U}, t, C)$.
- ($\dagger_2$) For some constant $\mathtt{Ext}_{\mathrm{No}}$, there are $\mathtt{Ext}_{\mathrm{No}}$ many possible extensions of C for each U. Each U submits $\min\{X_{\mathsf{U}}, \mathtt{Ext}_{\mathrm{No}}\}$ many requests from Λ (and only those) during timeslot t.

Let p_{U} be the probability that U is given permission to publish during timeslot t. In what follows, it will simplify calculations to consider what happens in the limit of the size of the network of users: We shall say that a given condition holds *in the limit*, if it holds so long as p_{U} is sufficiently small for all U. We'll say that one quantity x is proportional to another quantity y in the limit, if there exists some constant c such that, for each $\epsilon > 0$, $x/cy \in (1-\epsilon, 1+\epsilon)$ in the limit.

Proposition 5.1 below says that, when the number of possible extensions $\mathtt{Ext}_{\mathrm{No}}$ is larger than $X_{\max}$, the single-permitter O automatically gives rise to a PoW protocol, since, in the limit, the probability U is given permission to publish is then proportional to U's computational power. While

Proposition 5.1 works according to the specific assumption that the permitter responds to each request with independent probability, it should be clear that the basic principle holds under much more general conditions.

PROPOSITION 5.1. *Suppose that* $(\dagger_1)$ *and* $(\dagger_2)$ *above are satisfied, so that* O *is a single permitter, and each* U *submits* $min\{X_{\mathsf{U}}, \mathtt{Ext}_{No}\}$ *many requests during timeslot* t. *Let* p_{U} *be the probability that* U *is given permission to publish during timeslot* t. *In the limit,* p_{U} *is proportional to* $\mathcal{R}(\mathsf{U}, t, C) \cdot min\{X_{\mathsf{U}}, \mathtt{Ext}_{No}\}$.

PROOF. Define $Y_{\mathsf{U}} := \min\{X_{\mathsf{U}}, \mathtt{Ext}_{\mathrm{No}}\}$, so that U makes Y_{U} many requests during timeslot t. If $Y_{\mathsf{U}} = 0$ then $p_{\mathsf{U}} = 0$ and $\mathcal{R}(\mathsf{U}, t, C) \cdot Y_{\mathsf{U}} = 0$. So suppose otherwise. Let λ be as defined in $(\dagger_1)$. Then the probability that at least one of the Y_{U} many requests made by U results in permission to publish is $1 - (1 - \lambda \cdot \mathcal{R}(\mathsf{U}, t, C))^{Y_{\mathsf{U}}}$. It therefore suffices to show that:

$$\frac{1 - (1 - \lambda \cdot \mathcal{R}(\mathsf{U}, t, C))^{Y_{\mathsf{U}}}}{\lambda \cdot \mathcal{R}(\mathsf{U}, t, C) \cdot Y_{\mathsf{U}}} \to 1 \text{ in the limit.}$$

This can be shown with a straightforward analysis. Expanding out $(1 - \lambda \cdot \mathcal{R}(\mathsf{U}, t, C))^{Y_{\mathsf{U}}}$:

$$(1-\lambda\cdot\mathcal{R}(\mathsf{U}, t, C))^{Y_{\mathsf{U}}} = 1-\lambda\cdot\mathcal{R}(\mathsf{U}, t, C)\cdot Y_{\mathsf{U}}+\frac{1}{2}Y_{\mathsf{U}}(Y_{\mathsf{U}}-1)(\lambda\cdot\mathcal{R}(\mathsf{U}, t, C))^2-\frac{1}{6}Y_{\mathsf{U}}(Y_{\mathsf{U}}-1)(Y_{\mathsf{U}}-2)(\lambda\cdot\mathcal{R}(\mathsf{U}, t, C))^3+\cdots$$

We therefore have:

$$\frac{1 - (1 - \lambda \cdot \mathcal{R}(\mathsf{U}, t, C))^{Y_{\mathsf{U}}}}{\lambda \cdot \mathcal{R}(\mathsf{U}, t, C) \cdot Y_{\mathsf{U}}} = 1 - \frac{(Y_{\mathsf{U}} - 1) \cdot \lambda \cdot \mathcal{R}(\mathsf{U}, t, C)}{2} + \frac{(Y_{\mathsf{U}} - 1)(Y_{\mathsf{U}} - 2)(\lambda \cdot \mathcal{R}(\mathsf{U}, t, C))^2}{6} + \cdots . \quad (1)$$

Now, since $Y_{\mathsf{U}} > 0$, we have $\lambda \cdot \mathcal{R}(\mathsf{U}, t, C) \leq p_{\mathsf{U}}$, so that $\lambda \cdot \mathcal{R}(\mathsf{U}, t, C)$ must tend to zero as p_{U} tends to 0. Since Y_{U} is always less than the fixed bound $X_{\max}$, the r.h.s. of (1) therefore tends to 1 in the limit. □

In a standard PoS protocol, for example, one usually runs the lotteries choosing users to produce blocks by having users hash their public key, or some signed message, together with the timeslot identifier and a frequently updated 'random seed'. If the resulting hash (considered as a real number) is the lowest produced, or if it is below a threshold that depends on their stake, then that user might be allowed to produce the next block. If one wanted the permission to publish to be block-specific, one *could* require users to enter each proposed block as an extra input to the hash. Doing so would mean that users who intend to produce blocks are now incentivised to churn through many different possibilities for the block as entry to the hash. So the resulting protocol becomes a PoS/PoW hybrid.

In principle, however, and in situations where less possibilities are required for each block, one certainly can envisage protocols which use single-permitters, and which could be implemented using PoS. As a simplistic example, we might consider a protocol which is aimed at recording the time of a particular event. At each in a sequence of short timeslots, a single user might be selected and given permission of one of two forms. Either:

(a) They are given permission to publish a block recording that, "The event has happened by this timeslot", or;
(b) They are given permission to publish a block recording that, "The event is yet to take place".

To ensure single-permitting, the parent of the block should also be specified as part of the permission (e.g., with permission being given for different parents in some rotating fashion). Honest users are then asked to publish the permitted block only in the case that the information recorded by the block and all ancestors is correct. Such a protocol can be implemented using PoS, and the small number of possibilities for each block means that one can do so without degenerating into a PoW protocol.

6 DISCUSSION

6.1 Defining finality

The term 'finality' is sometimes used to mean the *absolute* guarantee that blocks of transactions will not be revoked once committed to the blockchain with a suitable level of confirmation. We have defined a different (probabilistic) notion of finality, and have argued that it can be effectively applied to the categorisation and analysis of blockchain protocols. It may be instructive, however, to further examine whether the former informal notion – let's call it *absolute finality* – is likely to be useful for the analysis of blockchain protocols.

To make things concrete, let us consider the case of Algorand. For the purposes of this discussion, all one needs to know about Algorand is that block confirmation revolves around the selection of committees, and that the protocol relies for its security on the idea that an adversary with suitably bounded stake will never have a committee majority.[10] Under appropriate modeling assumptions, one can show that the chance of the adversary gaining a committee majority at any point during the predetermined duration of the protocol is indeed negligible. Since the process of selecting users to be committee members is probabilistic, however, it certainly is *possible* that there will exist committees controlled entirely by the adversary. At a given moment in time it *could* turn out to be the case, even if only with negligible probability, that a number of prior committees have actually had dishonest majorities, and are now providing confirmation for an alternative blockchain. So Algorand fails to have absolute finality as a simple consequence of the fact that certain aspects of the process are best modeled as probabilistic.

The question then becomes, is it meaningful in a blockchain context to worry about the distinction between an event which occurs with probability which is *essentially* 0, and an event which holds with probability *exactly* 0? While it might be possible for a committee to have a dishonest majority, how much does this matter if the probability is $< 10^{-10}$ that this occurs at any time during the execution of the protocol? We take the position that if a permissionless protocol achieves absolute finality given appropriate modeling assumptions (such as the security of elliptic curve cryptography, or the fact that a given hash function is collision resistant), then it still holds with non-zero probability that some aspect of the modeling assumptions fails to hold. So the distinction is really a matter of where one hides the probability of failure.

6.2 Does finality matter?

The extent to which protocol finality is important is an interesting question. We have defined finality here so as to be most useful for classification purposes. The notion of finality that we consider requires being secure in the face of unbounded periods of network failure; one might argue that this is overkill in practice. For example, one relaxation would require only that a protocol be secure in the face of *realistically bounded* periods of network failure; this, in turn, may allow for greater protocol adaptivity.

Let us explore this idea further in the context of Bitcoin. In Section 2.8, we considered how to model a PoW protocol that was a simplified version of Bitcoin, in the sense that we did not consider the updates to the 'difficulty parameter' that are implemented every couple of weeks in Bitcoin. Now that we have formally defined security and adaptivity, we can consider in more detail what differences are caused by these updates to the difficulty parameter. In fact, Bitcoin is normally considered to be executed with a notion of confirmation which is particularly insensitive to the difficulty parameter – a block is considered confirmed once it belongs to the longest chain and is followed by a fixed number of blocks (six being a common choice). According to this notion of confirmation, network partitions of a few hours may suffice to produce a situation in which

[10] In fact, never more than a third of any given committee.

different blocks are confirmed for different users. If one wants to avoid this, one response is to consider the same protocol paired with a notion of confirmation that requires blocks to be produced at a certain *rate*. For example, one might consider a block to be confirmed if it belongs to the longest chain and is followed by $x \geq 6$ many blocks, which have been produced in less than $x/5.5$ many hours. The Bitcoin protocol with this notion of confirmation is still adaptive, but the network partitioning attack described in the proof of Theorem 4.1 would now have to be carried out over a considerably extended interval of time. One might argue that such extended network partitions are unlikely, and that, realistically speaking, adaptivity (even if slow) is likely to be beneficial in ensuring liveness.

7 CONCLUDING REMARKS

Our main aim in this paper has been to establish a framework for analysing permissionless blockchain protocols that blackboxes the precise mechanics of the user selection process. Establishing such a framework allows us to prove impossibility results, and to isolate the properties of the selection process which are significant in the sense that they impact the way in which the protocol must be designed, or influence properties of the resulting protocol, such as security in a range of settings. We have focussed on the difference between the sized and unsized settings, and have shown that the choice of setting is intimately related to a fundamental tradeoff for cryptocurrency protocols: A protocol cannot deliver finality for block confirmations while at the same time being adaptive. The formal dichotomy which results, can be seen as elucidating the informal division of permissionless blockchain protocols into those which are longest chain type protocols such as Bitcoin on the one hand, and those protocols such as Algorand, Casper FFG [5] or proof-of-stake (PoS) implementations of Tendermint or Hotstuff on the other, which work by importing traditional Byzantine-Fault-Tolerant protocols from the permissioned to the permissionless setting.

In the description of the framework presented here, explicit mention was made of an adversary who displays byzantine behaviour. The expectation is that properties of protocols are asserted modulo the existence of a bounded adversary. So assertions of liveness and security are made in a setting with explicit bounds on the adversary, and the requirement is that the protocol should behave well irrespective of the behaviour of the adversary, within the given bounds. This is an entirely standard form of analysis in the distributed computing literature. There is a general understanding in the blockchain community, however, that in the blockchain setting there is also the need for a deeper game-theoretic analysis, which takes account of user incentives. It is not enough that the protocol should perform well given arbitrary behaviour from the adversary. Given arbitrary behaviour by the adversary, it should also be the case that the instructions of the protocol constitute something like a Nash equilibrium for the honest users. It would be interesting to use and expand our framework in order to achieve impossibility results along these lines.

As well as allowing for impossibility results, a benefit of our framework may also be in providing some modularity for the description and analysis of protocols. For example, the description of PoS protocols tends really to consist of two components. One has to describe how lotteries are to be implemented securely, so as to provide an appropriate mechanism for user selection, and then one has to describe the protocol to be carried out by users who are selected to update the state. Once a mechanism for orchestrating lotteries has been agreed on (such as that used in Algorand), one might then want to describe a range of protocols, which work very differently from each other, but which use the same basic method of user selection. Or one might want to describe a protocol that uses the same method of user selection as Algorand, but which could be updated to use another method of user selection should something superior be developed later. Blackboxing the process of user selection via the use of permitters may therefore allow for a more modular description and analysis.

A further avenue for research would be to use the framework we have described here to formalise another notable difference between protocols which are adaptive and protocols which have finality, which concerns the nature of 'proof of confirmation'. For BFT protocols, it will generally be the case that the very existence of a certain set of signed objects may suffice to establish confirmation with high probability. For example, in Algorand, the existence of a block, together with an appropriate set of committee signatures establishing consensus for inclusion of the block, is sufficient to prove beyond reasonable doubt that the block can be considered confirmed. For Bitcoin and other adaptive protocols, on the other hand, a user will only believe that a certain chain is the longest until they are shown a longer chain. For the adaptive protocols, in other words, one needs to see a user's full published state in order to know whether they consider a given block to be confirmed. For protocols with finality, by contrast, certain sets of publications will constitute proof of confirmation, simply by virtue of being a *subset* of a user's state. We suspect that there are interesting interactions with the resource setting to be explored in this regard.

REFERENCES

[1] Iddo Bentov, Rafael Pass, and Elaine Shi. 2016. Snow White: Provably Secure Proofs of Stake. *IACR Cryptology ePrint Archive* 2016, 919 (2016).

[2] Jonah Brown-Cohen, Arvind Narayanan, Alexandros Psomas, and S Matthew Weinberg. 2019. Formal barriers to longest-chain proof-of-stake protocols. In *Proceedings of the 2019 ACM Conference on Economics and Computation.* 459–473.

[3] Ethan Buchman, Jae Kwon, and Zarko Milosevic. 2018. The latest gossip on BFT consensus. *arXiv preprint arXiv:1807.04938* (2018).

[4] Vitalik Buterin. 2018. What is Ethereum? *Ethereum Official webpage. Available: http://www. ethdocs. org/en/latest/introduction/what-is-ethereum. html. Accessed* 14 (2018).

[5] Vitalik Buterin and Virgil Griffith. 2017. Casper the friendly finality gadget. *arXiv preprint arXiv:1710.09437* (2017).

[6] Miguel Castro and Barbara Liskov. 2002. Practical Byzantine fault tolerance and proactive recovery. *ACM Transactions on Computer Systems (TOCS)* 20, 4 (2002), 398–461.

[7] Miguel Castro, Barbara Liskov, et al. 1999. Practical Byzantine fault tolerance. In *OSDI*, Vol. 99. 173–186.

[8] Jing Chen and Silvio Micali. 2016. Algorand. *arXiv preprint arXiv:1607.01341* (2016).

[9] Christian Decker and Roger Wattenhofer. 2013. Information propagation in the bitcoin network. In *IEEE P2P 2013 Proceedings*. IEEE, 1–10.

[10] Cynthia Dwork, Nancy A. Lynch, and Larry Stockmeyer. 1988. Consensus in the Presence of Partial Synchrony. *J. ACM* 35, 2 (1988), 288–323.

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

[12] Leslie Lamport, Robert Shostak, and Marshall Pease. 2019. The Byzantine generals problem. In *Concurrency: the Works of Leslie Lamport.* 203–226.

[13] Nancy A Lynch. 1996. *Distributed algorithms.* Elsevier.

[14] Satoshi Nakamoto. 2019. *Bitcoin: A peer-to-peer electronic cash system.* Technical Report. Manubot.

[15] Marshall Pease, Robert Shostak, and Leslie Lamport. 1980. Reaching agreement in the presence of faults. *Journal of the ACM (JACM)* 27, 2 (1980), 228–234.

[16] Yonatan Sompolinsky and Aviv Zohar. 2015. Secure high-rate transaction processing in bitcoin. In *International Conference on Financial Cryptography and Data Security*. Springer, 507–527.

[17] Maofan Yin, Dahlia Malkhi, Michael K Reiter, Guy Golan Gueta, and Ittai Abraham. 2018. Hotstuff: Bft consensus in the lens of blockchain. *arXiv preprint arXiv:1803.05069* (2018).