Permissionless Consensus in the Resource Model

Benjamin Terner

UC Irvine bterner@uci.edu

Abstract. This paper introduces a new model that abstracts resource-restricted distributed computation and permits simpler reasoning about consensus protocols in the resource-restricted regime. Our model introduces a simple abstraction – simply called "resources" – to capture a resource-restricted primitive which is general enough to capture most Proof of X such as Proof of Work and Proof of Stake. The supply of such resources is scarce, and a single resource allows a party to send a single message with elevated protocol status. For example, every puzzle solution in Proof of Work or Proof of Stake is a resource; the message associated with each resource is the payload of the puzzle. We show the power of resources for the problem of consensus, in which participants attempt to agree on a function of their inputs. We prove that given few additional assumptions, resources are sufficient to achieve consensus in the permissionless regime, even in the presence of a full-information adversary that can choose which parties get resources and when they get them. In the resource model, the participants do not need to know a bound on network delay, they do not need clocks, and they can join and leave the execution arbitrarily, even after sending only a single message. We require only a known upperbound on the rate at which resources enter the system, relative to the maximum network delay (without needing to know the network delay), and that over the long term, a majority of resources are acquired by honest participants. Our protocol for consensus follows from a protocol for graph consensus, which we define as a generalization of blockchains. Our graph consensus works even when resources enter the system at high rates, but the required honest majority increases with the rate. We show how to modify the protocol slightly to achieve one-bit consensus. We also show that for every graph consensus protocol that outputs a majority of honest vertices there exists a one-bit consensus protocol.

Keywords: Consensus, Blockchain, Permissionless, Resource Restricted

1 Introduction

The distributed system problem of *consensus*, in which participants in the protocol communicate over a network in an attempt to agree on a single bit or an append-only log based on their inputs, has been studied for decades since the seminal works of [18, 24]. The advent of Bitcoin [21] ushered in renewed interest in consensus protocols by introducing the *permissionless* regime. The permissionless regime models internet-scale protocols in which participation is *dynamic*, meaning participants can join and leave an execution arbitrarily, the number of active participants may be in constant flux, the identities of the participants at any point in time are unknowable, and the adversary may control arbitrarily many parties.

Consensus protocols for the permissionless regime have proliferated since Bitcoin [5, 13] as new approaches have focused on the resource-restricted model. The resource-restricted model changes the basis of security from the proportion of honest participants in a system to the physical resources that they control. In Eurocrypt 2020, Garay et al [12] formalized a randomized resource-restricted model and showed that by restricting the ability of parties to send messages, it is possible to bypass known bounds for both Byzantine Agreement and MPC. Bitcoin famously requires participants to solve Proof of Work to participate [2, 10, 21]. In response to Proof of Work's wasteful computation, many Proof of X (PoX) variants have been proposed (see [5]), the most popular being Proof of Stake [3, 8, 16] (PoS).

Resources: A Unifying Abstraction for a New Model To better understand the permissionless regime and the power of PoX, we ask:

Is there a unifying abstraction for PoX that implies consensus in the permissionless regime? If so, under what assumptions does it imply consensus?

In this work, we model a unifying abstraction of PoX which we simply call *resources.* We use resources to cast permissionless protocols into a *new model* in which a subset of messages are given special elite status, and the supply of these messages are constrained. We model the "competition process" for resources implicitly (and more generally) by deferring it to the environment. This allows us to decouple the resource-producing process (usually, but not limited to, mining) from the resource-consuming process (in our case, a graph protocol). We then isolate a few properties of resources, and show that any arbitrarily bad resource-producing process (implemented by the environment, or forwarded to our protocol by the environment) allows us to achieve consensus, as long as resources satisfy the properties we present. We show that it is possible to achieve consensus if (a) a majority of resources are received by honest parties over the long term, and (b) an upper bound on the rate at which resources enter the system is known.

Our novel protocol requires weaker synchronization assumptions than current practice; this shows that we may weaken the assumptions required for consensus by building PoX from weaker synchronization assumptions. To date, all other works we know for the permissionless model require either knowledge of the network delay [7, 8, 11] or (weakly synchronized) clocks [3, 4, 9, 16], plus some assumption about the number of active participants. In our model, parties have no way to synchronize, they may join and leave an execution arbitrarily, and there is no bound on the number of parties in an execution.

Specifically, we make the following contributions:

1. We provide a simple abstraction of PoX called *resources* with very few properties. A resource is a black-box generalization of a puzzle solution, which holds a string chosen by the party that "discovers" the resource.
2. We show that our abstraction implies consensus while requiring weaker assumptions and facing a stronger adversary than existing constructions.
3. We argue that the uses of PoX for consensus (that we know of) implement our abstraction, and that our model generalizes current designs.

Generalizing Resource Generation Resources can be implemented in many ways that are not limited to cryptographic puzzles; any process that achieves the properties we describe can be used to achieve consensus. As a thought experiment, we consider that before a protocol execution, some setup may select specific parties to be designated as "leaders" at specific points in the execution. (For example, an execution could be divided into epochs, and during each epoch a small set of leaders may be chosen.) During the execution, those parties are informed that they are leaders and permitted to send a single message to all other parties, along with a certificate that they have been selected as a leader at that moment. (Looking ahead, we will show that this leader selection can even be determined adaptively by the adversary.) We defer a discussion of generalized resource generation to the full version [25].

1.1 Overview of Our Model

The Permissionless Regime We give a short overview of our model here, and a full syntactic framework in the full version [25]. An execution proceeds in rounds and is directed by the environment, which serves as the adversary. Participation is dynamic (meaning parties can join and leave the execution arbitrarily), and completely controlled by the adversary; in every round, the adversary controls which parties send and receive messages, and which are completely inactive. Parties that send or receive messages in a round are considered *active* in that round. The maximum network delay Δ is unknown and participants do not have clocks. The number of participants is unknown to the parties *and may be unbounded.* Moreover, the adversary has *full information* about the states of all honest parties; it can corrupt parties adaptively; and it chooses which parties receive resources and when they receive them. (Note that in the full information model, we do not have digital signatures; this work shows that given resources, we do not additionally need signatures to imply consensus.) Communication occurs over peer-to-peer channels or via multicast. Honest parties cannot tell whether a message was sent over a peer-to-peer link or over multicast (this allows corrupt parties to selectively send messages to some parties but not to others).

Any protocol that is secure in our model must achieve consensus even when every honest participant sends *at most one message* before it leaves the execution, and even when every honest participant is only active for a (very) short period of time from the moment it joins to the moment it leaves. (In the extreme, just long enough to receive the state of the system and send a single message.)

The Properties of Resources Recall that in our model, the environment both directs the execution and abstracts the resource-producing process. Therefore, rather than requiring participants to solve a PoX puzzle, we say that participants are *allocated* resources from the environment, and our protocols show how participants use the resources they receive. We express properties of resources that mimic PoX via a syntactic model, in which the properties of resources are expressed as constraints on an execution transcript. The full model is in the full version [25]; we overview the syntactic model and properties here.

We model resources as a set of symbols Ψ, and require that a protocol specify how parties respond when they receive resources. When a resource $\psi \in \Psi$ is allocated by the environment to a party p, a special event called a *resource allocation* is recorded in the transcript, which states that p receives ψ. The following rules govern how these symbols may appear in the transcript.

1. **Unforgeability** No participant can "fake" the fact that it has a resource. In practice, PoX schemes enforce this requirement by requiring that PoX solutions must be found by solving some puzzle, and the solutions are verifiable by other participants. Formally, a transcript satisfies *resource unforgeability* if no resource appears in the transcript before its allocation event. This enforces that parties are constrained to obtaining resources *only* by receiving them from the environment, which abstracts the resource-producing process.
2. **Binding** Each resource can be *bound* with one and only one string, which gives the resource semantics. The string must be chosen at the moment that the resource is generated. This models that in PoX schemes, parties attempt to solve puzzles with respect to a specific message they wish to send. (In some implementations, this message includes a public key that boostraps special status to future messages signed with that key.) Formally, a string m bound to a resource ψ is encoded as $\psi||m||\psi$ [1], where $||$ denotes concatenation. A transcript satisfies *resource binding* if for any two encodings $\psi||m||\psi$ and $\psi'||m'||\psi'$: $\psi = \psi'$ implies $m = m'$.

Constraining the Supply of Resources We model constraints on resources which mimic the assumptions common to PoX mechanisms:

1. **Long Term Honest Majority** Over any period of time in which n resources are allocated, we require that $\alpha n - \varepsilon$ are allocated to honest participants and at most $\beta n + \varepsilon$ are allocated to corrupt participants, where $\beta = 1 - \alpha$. When $\alpha > \beta$, we say that honest participants receive a *long term* majority of resources. ε represents a short-term corrupt advantage, which models an adversary which pools its physical resources in order to achieve a short "burst" of resources.
2. **Rate Limit** We let ρ upperbound the number of resources that may be generated per Δ time, where Δ is the (unknown) maximum network delay.

[1] This is a standard encoding technique. By encompassing the message with its resource, it is clear where the string bound to the resource begins and ends

In the full version [25], we give a full treatment explaining how Proof of Work and Proof of Stake implement resources, including what constitutes a resource for each scheme. We stress that in most designs, *every solution of a cryptographic puzzle is a resource.* We additionally explain how the model captures other cryptographic and non-cryptographic PoX.

1.2 Main Results

One-Bit Consensus Our main result is that resources imply consensus in our new model. In the consensus problem, all parties have an input $b \in \{0, 1\}$, and they must output some value $v \in \{0, 1\}$ subject to the following constraints. By *agreement*, all parties must output the same bit. By *nontriviality*, if every honest party has the same input, then they must output that bit. By *termination*, the protocol must terminate after a finite number of resources have been allocated. We show that resources imply consensus assuming only knowledge of ρ, which upperbounds the rate at which resources are allocated relative to the (unknown) maximum network delay, and that honest parties receive a (large enough) majority of resources in the long term.

Theorem 1 (Informal). *Let $c = O(\rho + \varepsilon)$. For all $\alpha > \rho c(1 - \alpha)$, there exists a one-bit consensus protocol in the permissionless regime with resources.*

Graph Consensus We build one-bit consensus from resources using a technique reminiscent of so-called blockchains. We define a problem called graph consensus in which honest participants maintain local graphs and propose vertices to be included in each other's graphs. The security goals of a graph consensus protocol are generalizations of those proposed by [6, 14, 22]. A graph consensus protocol should achieve two properties. *Consistency* requires that for any two graphs output by honest participants, one participant's output must be a subgraph of the other. *Liveness* requires that honest participants may not trivially output empty graphs, but that their outputs grow over time.

Theorem 2 (Informal). *Let $c = O(\rho + \varepsilon)$. For all $\alpha > \rho c(1 - \alpha)$, there exists a graph consensus protocol in the permissionless regime with resources.*

Notably, we show that it is possible to achieve graph consensus when $\rho > 1$, i.e. more than 1 resource may allocated per Δ time. However, interestingly, our protocol requires that α grow with $O(\rho^2(1 - \alpha))$ in order to maintain security.

Necessity of Assumptions For completeness, we additionally show the necessity of our assumptions. The proof of the following theorem follows from standard techniques, and the discussion is deferred to the full version [25].

Theorem 3 (Informal). *There is no consensus protocol in the permissionless model that does not require both a long-term majority of resources and a constraint on the network delay.*

The Parameter Regime The parameter regimes for which our protocols are secure are not competitive with existing designs. For example, the protocols are secure for $\alpha = 0.865, \epsilon = 1, \rho = 1$, or $\alpha = 0.954, \epsilon = 2, \rho = 2$, but these are not comparable to the best parameters for protocols which make stronger assumptions. Nevertheless, we provide feasibility results in the presence of a very strong adversary. Our regime is overly restrictive, as we comment below that no longest-chain protocol can be proven secure for nontrivial rates ($\rho > 1$). However, the fact that consensus is achievable even in such a difficult regime is a stronger statement to the power of resources.

1.3 Technical Overview

Our technique to build consensus builds directly on our graph consensus protocol. We show that given a long-term majority of resources and a bound on the rate, honest participants can use the properties of resources to build a directed acyclic graph (DAG) which captures the (partial) ordering in which they receive their resources. Importantly, every vertex in the global DAG is associated with a resource (much like every vertex in a blockchain is associated with a PoX). The unforgeability and binding properties of resources enforce that corrupt participants cannot manipulate the graph structure. The honest participants embed structure into the graph that can be used to infer when corrupt parties attempt to cheat by "withholding" their resources, i.e. not immediately multicasting a vertex they have added to the graph.

In our graph protocol, we use the long term honest advantage in resources similarly to many longest-chain blockchains. We define the depth of a vertex in a DAG to be the length of the longest path from the root to that vertex (where the DAG grows from a root with no indegree to the leaves with no outdegree). We then require that the honest participants can build deeper branches on the DAG than the corrupt participants.

The structure that honest participants build into the global DAG is *reachability*. Every honest vertex which is added to the global DAG is guaranteed to gain an honest successor, and to always be a predecessor of one of the deepest vertices in the global DAG. However, corrupt vertices are not guaranteed to become predecessors of any honest vertices. If honest participants can build longer paths in the global DAG over time than corrupt participants, then if corrupt participants withhold their vertices for too long, their withheld branches will eventually fall behind the depth of the global DAG. Honest participants extract their outputs by selecting vertices in their local views of the global DAG which are predecessors of the deepest vertices in their views, excising all corrupt vertices on branches which have fallen short. Our technical challenge is to compute how long it takes – measured in depth – for a withheld branch to fall short of the honest parties' branch.

One-bit consensus follows from any graph consensus protocol which guarantees that for any sufficiently large output graph, a majority of the vertices must be associated with resources allocated to honest participants.

Why Chain Protocols Fail: Pathological Chain Structures In our model, *no longest-chain or heaviest-chain protocol can be proven secure* at non-trivial resource rates ($\rho > 1$). Consider an execution of a chain protocol in which a fork develops at the root and is never resolved. Because in our model, when $\rho > 1$ the adversary can always allocate multiple resources concurrently, forks in chain protocols can be perpetuated indefinitely. Therefore, although a party's local graph grows as a function of the number of resources that have been allocated, consistency requires that no party can ever output either branch of the fork. In this case, liveness fails because no party ever outputs any vertices. Note that this may happen even if the corrupt participants receive no resources. In comparison, in a random model, the random distribution of resource arrivals implies that forks will be resolved eventually, which allows participants to eventually output one branch. The perpetual fork attack is also discussed in [17].

1.4 Related Work

Comprehensive overviews of the blockchain literature can be found in the systemizations of knowledge by [5], [13], and [26](who introduced the term PoX). Here we describe only works we know about that address the properties of resource-like objects; in the full version [25] we give extended related work on the permissionless model and consensus protocols.

Generalizing Resource-Constrained Results This work generalizes the findings of other works, surprisingly showing that impossibility results that depend on strong assumptions need not hold if another system parameter can be bounded. The work of Lewis-Pye and Roughgarden [19], which originally appeared online after this work but has related themes, proves a CAP-style theorem that a protocol cannot be secure in the partially synchronous setting when the size of the resource pool is unknown. Similarly, Pass and Shi [7, 23] prove that for protocols which require mining, if the maximum network delay is unknown then the number of participants must be known within a factor of 2, even when participants are synchronous and have clocks. Intuitively, the number of participants are proxy for the mining rate; in the attack, the adversary splits the execution into two groups, and delivers messages within each group quickly but between groups slowly. These works implicitly assume that the mining rate cannot be bounded without the assumptions in their models. Our work show that *an upperbound on the rate* of resources relative to the network delay (in the above cases, puzzle solutions) is a sufficient network assumption; if this can be approximated without granular knowledge of the above required system parameters, consensus is still possible. Therefore, we show that is possible to achieve consensus in an expanded set of environments where the resource rate can be upperbounded. (For discussions on deferring resource generation to the environment, and on why a known upper bound on the rate is a weaker assumption than previously studied, refer to the full version [25]).

Properties of PoX As far as we know, no other works present the common qualities of PoX via a single abstraction. However, Miller et al [20] model Proof of Work as scratch-off-puzzles, showing a number of desirable properties for Proof of Work objects. Alwen and Tackman [1] model desirable properties for moderately hard puzzles. Garay et al. [15] model the sufficient properties of PoW to yield consensus. Garay et al. [12] further abstract the properties of PoW to a randomized resource-restricted model.

1.5 Paper Organization

In Section 2 we define graph consensus in our model. In Section 3 we present our main protocol, our main theorem, and an overview of the proof. In the full version [25], we include the following discussions: We discuss how several popular forms of PoX implement resources. We discuss our modeling choices and frame our results with respect to other models; we include discussions of whether it is reasonable to know the resource rate, and why knowing an upper bound on the rate is weaker than knowing the network delay. We provide our full formal model based on a syntactic framework for resources. We prove security of our graph consensus protocol. We define one-bit consensus in our model, provide a protocol that achieves it, and provide a generic transformation from graph consensus to one-bit consensus. We prove that honest majority and some bound on the network are necessary for consensus in the permissionless regime.

2 Graph Consensus Problem

2.1 Preliminaries for Graphs

A graph $G = (V, E)$ is a set of vertices and a set of edges between vertices. For a graph G, we denote the set of its vertices as $G.V$ and its edges as $G.E$. In this work we consider only directed acyclic graphs (DAGs); we therefore use term graph to refer to a DAG. A *root vertex* in a graph is a vertex with in-degree 0. In this work, every graph which we consider has exactly one root vertex, which in cryptocurrencies is also called a genesis vertex.

We define depth of a vertex and depth of a graph in a non-standard way:

Definition 1 (Depth of a Vertex, Depth of a Graph). *Let* root *be the root vertex of a graph G. The* depth *of a vertex v in G is defined as the length of the* longest *path from* root *to v. The* depth *of G is defined as the depth of its deepest vertex.*

We use $\mathsf{D}(G)$ to denote the depth of a graph G, and use $\mathsf{D}_G(v)$ to denote the depth of a vertex v in G. When the graph is implied from context, we simply write $\mathsf{D}(v)$. The depth of a root vertex is always 0. We use $G|_d$ to denote the subgraph of G including only vertices with depth $\leq d$. Figure 1 illustrates the depths of vertices in a simple graph. We denote a path from vertices v to u as $v \to u$. A path $v \to u$ *spans d depth* if $\mathsf{D}(u) - \mathsf{D}(v) = d$. We say $u \in G.V$ is

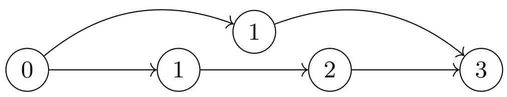

Fig. 1. An example graph in which each vertex is labeled with its depth. The root vertex has depth 0 by definition, and every other vertex's depth is defined by the longest path from the root to the vertex.

reachable from $v \in G.V$ if there is a path $v \to u$. For a vertex $v \in G.V$, the *predecessor graph* of v is the subgraph of G containing v and every vertex and edge on every path from root to v. We use $\cup$ to denote graph union and $\subseteq$ to denote a subgraph. We let $\mathsf{indegree}(v)$ denote the indegree of a vertex v and $\mathsf{outdegree}(v)$ denote its outdegree. (In the full version [25], a vertex may have a "payload" string that gives semantics to the vertex.)

2.2 Graph Consensus Protocol

In an execution of a graph consensus protocol, participants have no input. Each participant p maintains a local graph G_p based on the messages it has received so far and the protocol specification. A graph consensus protocol specifies how participants generate new vertices, and how to propose that other participants include the new vertices in their local graphs. It also specifies how a participant determines whether a new vertex, which it receives in a proposal from another participant, should be included in its local graph. For a participant p active at time t, we denote by $G_p^{(t)}$ its local graph after all vertices are added at t. Each participant p additionally maintains an output graph G_p^*, which it outputs whenever it is active. The protocol must specify a deterministic way for each p to compute G_p^* as a function of its local graph G_p. We denote by $G_p^{*(t)}$ the output of p at time t.

An execution of graph consensus may continue indefinitely. The goal of a protocol is for the participants' outputs to obey consistency and liveness properties across time. Graph consistency requires that if participants p active at t and q active at t', output $G_p^{*(t)}$ and $G_q^{*(t')}$, then one output graph must be a subgraph of the other.

Definition 2 (Graph Consistency). *An execution satisfies* graph consistency *if for all times t and t', and for all honest p and q active at t and t', respectively: $G_p^{*(t)} \not\subseteq G_q^{*(t')} \implies G_q^{*(t')} \subseteq G_p^{*(t)}$.*

A protocol can trivially satisfy graph consistency if participants always output the empty graph. We therefore define liveness to require that each participant p's output G_p^* grows as a function of the number of resources which have been allocated up to some point in time, as follows:

Definition 3 (f-Liveness). *Let $f\colon \mathbb{N} \to \mathbb{N}$. An execution satisfies* f-liveness *if for every time t and honest participant p active at t: if the environment has allocated N resources by time t, then $|G_p^{*(t)}.V| \geq f(N)$.*

When $f = 0$, liveness is trivial because parties may always output the empty graph. When f is nontrivial we require a nontrivial protocol. Our protocols satisfy liveness for nontrivial f. We remark that unlike other definitions of liveness, ours does not require that a party's output graph grow as a function of time. Rather, we require that a party's output grow as a function of the number of resources allocated by the environment. Looking ahead, consider that a protocol which depends on resources should not need to make progress if there are no resources allocated. In our model, the fact that resources are produced depends on the environment, and we do not assume a lowerbound on the resource rate; however, only when the environment delivers sufficiently many resources are our protocols required to produce output.

In some applications, it is desirable to show that some proportion of the vertices in an honest participant's output must be generated by honest participants. If a vertex is generated by an honest participant, we call it an honest vertex; otherwise, we call it a corrupt vertex. We let $\mathsf{hon}(G.V)$ denote the honest vertices in G. We define h-honest-vertex liveness to quantify the guaranteed proportion of honest vertices in a participant's output graph.

Definition 4 (h-Honest-Vertex Liveness). *Let $h\colon \mathbb{N} \to \mathbb{N}$. An execution satisfies* h-honest-vertex liveness *if for every time t and honest participant p active at t:* $|\mathsf{hon}(G_p^{*(t)}.V)| \geq h(|G_p^{(t)}.V|)$.

In the rest of the paper, we refer to f-liveness and h-honest-vertex liveness together by f, h-liveness to say that a protocol satisfies both f-liveness and h-honest-vertex liveness.

3 Main Protocol

3.1 Protocol Description

Protocol Π^G, presented in Figure 2, is a graph consensus protocol. It is parameterized by α and ε, which describe the proportion of honest resources which are allocated, and the maximum rate of resource allocation ρ.

Each participant p maintains a local DAG G_p in which every vertex except the root is a resource. The graph G_p is initialized to $(\{\mathsf{root}\}, \emptyset)$, and grows from the root toward high depths throughout the execution as participants are allocated resources and receive messages. Whenever p is allocated a resource, it adds the resource to its graph as a new vertex, and then immediately multicasts its local graph including the new vertex to all honest participants. When an honest participant receives a message containing a graph, it updates its local graph to include new vertices and edges not previously in its local graph. We must show how a participant p chooses the predecessors of each vertex that it adds to its graph, and p computes its output G_p^* from its local graph G_p.

We describe resources as vertices as follows. When any participant is allocated resource ψ, we let v_ψ denote the vertex corresponding to ψ. When describing an arbitrary vertex, we denote it as v or u, eliding its respective resource.

When any honest participant p adds a new vertex to its graph, it adds the vertex to its graph as the new deepest vertex. Specifically, when p is allocated a resource ψ and adds vertex v_ψ to its local graph G_p, p adds an inbound edge to v_ψ from every vertex u in G_p which (a) has no outbound edges in G_p, and (b) is close in depth to G_p. When p is allocated ψ, it must also choose v_ψ's edges *immediately*, as p must bind the inbound edges of v_ψ to ψ. Because each vertex's inbound edges are bound to the vertex's respective resource, it may not gain additional predecessors.

Over time, some vertices will gain successors and some vertices may be "orphaned" and stop gaining successors. Each participant computes its output G_p^* as a subgraph of its G_p consisting of vertices which are both far from the end of its graph (measured in the difference in depth between the vertex and the graph) and are still gaining successors.

Encoding a Graph Using Resources We model a resource as a black box object which is *bound to a string* that conveys its semantics *at the moment* it is allocated. In Π^G, the string bound to each resources encodes the direct predecessors of its respective vertex; when a participant is allocated a resource ψ, it binds to ψ the encoding of each vertex which has an outbound edge to v_ψ. If no edges are bound to ψ, then v_ψ is defined to have an edge from root. In this way, each vertex is uniquely committed to its predecessors at the moment it is allocated. A participant multicasts its local graph by sending all of the bound resources which encode the vertices and edges in its local graph.

Event Responses We now detail how participants respond when they are allocated resources and when they receive messages, and we explain how participants compute their outputs from their local graphs.

On Resource Allocation When an honest participant p is allocated a resource ψ, we say that it *generates* a vertex v_ψ that it adds to its local graph G_p. Participant p chooses the inbound edges of v_ψ based on its current graph G_p by adding an edge to v_ψ from each vertex u in G_p for which both $\mathsf{outdegree}(u) = 0$ and $\mathsf{D}(G_p) - \mathsf{D}(u) < c$, where c is a constant computed from the protocol parameters and is the maximum depth spanned by an honestly chosen edge. Immediately after generating v_ψ, p multicasts its entire local graph containing v_ψ and its inbound edges.

On Receipt of a Message Every message sent between participants is an encoding of a graph. (Any other message is ignored.) When a participant p receives a graph G' in a message, it verifies that G' is a valid graph. If G' is valid, then p updates its local graph as $G_p \leftarrow G_p \cup G'$. If G' is not valid, then p ignores G'.

G' may be invalid in two ways. First, G' may contain an edge (v, u) which spans more than c depth. Second, G' may be "missing a vertex," meaning there is a vertex v in $G'.V$ for which not all of v's predecessors are in $G'.V$. (This means the graph G is incomplete in the party's view.)

Protocol 1 DAG Protocol for Graph Consensus $\Pi^G(\alpha, \varepsilon, \rho)$

```
Parameters: $\alpha, \varepsilon, \rho$
Derived Constants:
  1. $\beta = 1 - \alpha$
  2. $\gamma = (1+\beta)\rho + \varepsilon + \frac{\varepsilon}{\rho} + 1$
  3. $c = \gamma + \rho + \frac{\varepsilon}{\alpha}$
  4. $\ell_1 = \gamma + \rho$
  5. $\ell_2 = c(\varepsilon+1) + \rho + \frac{c\beta}{\frac{\alpha}{\rho} - c\beta}(c(\varepsilon+1) + (2+\beta)\rho + \frac{\varepsilon}{\alpha} + 2\frac{\varepsilon}{\rho} + 2)$
  6. $\ell^* = \ell_1 + \ell_2$
Internal Variables:
  1. $G_p = (V_p, E_p)$ is a participant's local state. Initially, $G_p = (\{\mathsf{root}\}, \emptyset)$
  2. $G_p^* = (V_p^*, E_p^*)$ is a participant's output graph. Initially, $G_p^* = (\emptyset, \emptyset)$
Event Responses:
  1. On Receiving a Graph $(G')$
     – $G_p \leftarrow G_p \cup \mathsf{validateGraph}(G')$
     – $G_p^* \leftarrow \mathsf{extract}(G_p)|_{\mathsf{D}(G_p)-\ell^*}$
  2. On Being Allocated a Resource $\psi$
     – $G_p \leftarrow \mathsf{addVert}(G_p, \psi)$
     – multicast $G_p$
     – $G_p^* \leftarrow \mathsf{extract}(G_p)|_{\mathsf{D}(G_i)-\ell^*}$
Internal Functions:
  1. $\mathsf{addVert}(G, \psi)$:
     – $V' \leftarrow \{u \in G.V \colon \mathsf{D}(G) - \mathsf{D}(u) < c$ and $\mathsf{outdegree}(u) = 0\}$
     – return new graph $G'$ such that
         • $G'.V \leftarrow G.V \cup \{v_\psi\}$
         • $G'.E \leftarrow G.E \cup \{(u, v_\psi) \colon u \in V'\}$
  2. $\mathsf{extract}(G)$:
     – $S \leftarrow \{v \in G.V \colon \mathsf{D}(G) - \mathsf{D}(v) \le c + \rho\}$ // "starting vertices"
     – return $S \cup \{v \in G.V \colon \exists u \in S$ such that $u$ is reachable from $v\}$
  3. $\mathsf{validateGraph}(G')$:
     – if
       (a) $\exists (u, v) \in G'.E$ such that $\mathsf{D}(u) - \mathsf{D}(v) > c$, or
       (b) $\exists (u, v) \in G'.E$ such that $u \notin G'.V$
       then return $(\emptyset, \emptyset)$
     – return $G'$
```

Fig. 2. Protocol Π^G for graph consensus

Computing Output An honest participant p computes its output G_p^* from its local graph G_p by first extracting a subgraph of G_p into an intermediate graph, and then outputting all but the deepest vertices in the intermediate graph. More precisely, p extracts a subgraph of G_p using the procedure $\mathsf{extract}(G_p)$, as follows. First, p selects a set of "starting vertices" as the set $S = \{v \in G_p \colon \mathsf{D}(G_p) - \mathsf{D}(v) < c + \rho\}$. Next, p extracts every starting vertex and every vertex from which any starting vertex is reachable. Finally, p outputs $G_p^* \leftarrow \mathsf{extract}(G_p)|_{\mathsf{D}(G_p)-\ell^*}$, which contains all the vertices in its extracted subgraph with depth less than $\mathsf{D}(G_p) - \ell^*$, where ℓ^* is derived from the protocol parameters.

Remark 1 (Sending a Whole Graph). Whenever a participant generates a new vertex, it multicasts its entire graph. We admit it is unrealistic in practice to multicast an entire local graph. Our protocol should be considered only theoretical. It remains future work to show that participants need not multicast their entire graphs whenever they generate a new vertex.

3.2 Theorem Statement

We now state our main theorem, which is that protocol Π^G satisfies graph consensus for appropriate parameters.

Theorem 4. *For all N, all ρ, and all ε, and for all $\alpha > \rho(1-\alpha)((3-\alpha)\rho + \frac{\varepsilon}{\alpha} + \frac{\varepsilon}{\rho} + \varepsilon + 1)$ every (α, ε)-honest, ρ-rate-limited, admissible execution of $\Pi^G(\alpha, \varepsilon, \rho)$ satisfies graph consistency and f, h-liveness for $f(N) = h(N) = \alpha N - \varepsilon - \rho(\ell^* + 1)$, where ℓ^* is a derived constant defined as in the protocol.*

Recall that in Π^G, each participant computes its output by extracting a subgraph from its local graph and then chopping off the deepest vertices in the extracted subgraph, where the chop-off threshold is the derived constant ℓ^*. Intuitively, liveness follows from the fact that as a participant's local graph increases in depth, the depth of the graph which it outputs also increases. The main objective of the proof is to show that the protocol achieves graph consistency.

The main desideratum of the proof of graph consistency follows:

Proposition 1. *Let $c = (3-\alpha)\rho + \frac{\varepsilon}{\alpha} + \frac{\varepsilon}{\rho} + \varepsilon + 1$ (as in Protocol Π^G). If $\alpha > \rho\beta c$, then for all k, times t and t', and honest participants p and q active at t and t', respectively, if $\mathsf{D}(G_p^{(t)}) > k + \ell^*$ and $\mathsf{D}(G_q^{(t')}) > k + \ell^*$, then $\mathsf{extract}(G_p^{(t)})|_k = \mathsf{extract}(G_q^{(t')})|_k$.*

where c and ℓ^* are defined as in the protocol.

Graph consistency follows directly from assigning $G_p^* \leftarrow \mathsf{extract}(G_p)|_{\mathsf{D}(G_p)-\ell^*}$, since when two honest participants output graphs, then the less deep output graph must always be a subgraph of the deeper (if the output graphs have the same depth, then they must be the same graph).

3.3 Proof Overview

We now overview the proof of Proposition 1. The full proofs of Proposition 1 and Theorem 4 are in the full version [25].

Building a Virtual Global Graph We consider that the participants collectively build a virtual global graph $\mathbb{G}$ throughout an execution. When the execution begins, $\mathbb{G}$ is initialized to a graph with only a root vertex. Whenever *any* participant is allocated a resource, the vertex that it generates is immediately added to $\mathbb{G}$. In particular, even if a corrupt participant generates a vertex and "withholds" the vertex by not sending it to any honest participant, the vertex is still added to $\mathbb{G}$ at the moment that it is generated. We denote by $\mathbb{G}^{(t)}$ the state of $\mathbb{G}$ after all vertices are added at time t.

$\mathbb{G}$ represents the global state of the execution. Consider that $G_p^{(t)}$ is p's its local view of $\mathbb{G}^{(t)}$, and it is easy to see that $G_p^{(t)}$ must be a subgraph of $\mathbb{G}^{(t)}$. Moreover, for every vertex $v \in \mathbb{G}^{(t)}.V$, if v is in $G_p^{(t)}$, then $\mathsf{D}_{\mathbb{G}^{(t)}}(v) = \mathsf{D}_{G_p^{(t)}}(v)$. Henceforth, when we refer to the depth of a vertex, we simply write $\mathsf{D}(v)$ because its depth is uniquely defined.

Outputting Predecessors and Omitting Orphans Recall that an honest participant p active at time t outputs a vertex v from its local graph $G_p^{(t)}$ if and only if $v \in \mathsf{extract}(G_p^{(t)})|_{\mathsf{D}(G_p^{(t)})-\ell^*}$. By applying $\mathsf{extract}()$ and chopping off the deepest vertices, the protocol enforces two requirements in order to output a vertex. First v must be far from the end of a participant's graph $(\mathsf{D}(G_p^{(t)}) > \mathsf{D}(v) + \ell^*)$. Second, v must be a predecessor of one of the starting vertices in $G_p^{(t)}$.

Intuitively, one can consider that every participant p decides whether each vertex v in its view should be output or not. However, p "waits" before making a decision until v is sufficiently far from the end of its graph. At that point, p does not output v only if v has been "orphaned." A vertex is "orphaned" if it is more than ℓ^* depth from the end of a graph but not a predecessor of one of the graph's starting vertices.

To achieve graph consistency, p must make the same decision on v as every other honest participant. We show that by the time the depth of G_p exceeds ℓ^* more than the depth of v, v's status as an orphan or not an orphan has been determined in $\mathbb{G}$ and will not change; moreover, v's orphan status in G_p must mirror its status in $\mathbb{G}$. If v is not a predecessor of one of the starting vertices in G_p, then v will never be a predecessor of a starting vertex in any honest participant's local graph which is deep enough to decide on v. However, if v is a predecessor of one of the starting vertices in G_p, then v will never be orphaned in any honest participant's local graph.

Consistency of Honest Vertices We first show consistency of the honest vertices which honest participants output. We do so by showing that no honest vertex is ever orphaned, and therefore *all* honest vertices are eventually output by honest participants. Our high-level lemma towards this statement actually says something stronger. It says that every honest vertex in $\mathbb{G}$ which is more than $\ell_1 < \ell^*$ distance from the end of an honest participant's graph must be extracted from the graph when it computes its output from its local graph.

Lemma 1 (Honest Vertex Extraction). *For every time t, honest participant p active at t, and honest vertex $v \in \mathbb{G}^{(t)}$: $\mathsf{D}(G_p^{(t)}) - \mathsf{D}(v) > \ell_1 \implies v \in \mathsf{extract}(G_p^{(t)})$.*

Lemma 1, consistency of honest vertices in participants' outputs, follows trivially from composition of Lemmas 2 and 3, described below. Lemma 2 shows that by the time $\mathsf{D}(G_p) > \mathsf{D}(v) + \ell_1$ for any honest participant's graph G_p and honest vertex v, enough time must have passed since v was originally multicast that v is in G_p. Lemma 3 shows that every such honest vertex in an honest participant's graph must be a predecessor of a starting vertex in the graph.

Consistency of Honest Vertices in Honest Views For the first step, we show that if an honest participant's local graph G_p is deeper than an honest vertex v by more than a fixed distance ℓ_1, then $v \in G_p$.

Lemma 2 (Depth-Based Indicator for Honest Vertices). *For all t, honest p active at t, and honest vertex $v \in \mathbb{G}^{(t)}$: $\mathsf{D}(G_p^{(t)}) - \mathsf{D}(v) > \ell_1 \implies v \in G_p^{(t)}$.*

Intuitively, ℓ_1 is derived as follows. Let t_v be the time that some honest vertex v is generated by honest participant q. Naively, one would like to claim that if $\mathsf{D}(G_p^{(t)}) - \mathsf{D}(v) > \rho$, then ρ vertices must have been generated after v, and it follows from the rate limit on resource allocations that $t > t_v + \Delta$. However, the naive attempt makes the unfounded assumption that at t_v, v must be the deepest vertex in $\mathbb{G}^{(t_v)}$. Instead, we derive a constant γ that gives the maximum difference between $\mathbb{G}^{(t)}$ and an honest view $G_p^{(t)}$ at any time t. We then derive $\ell_1 = \gamma + \rho$ and show that if $\mathsf{D}(G_p^{(t)}) - \mathsf{D}(v) > \ell_1$, then Δ time must have elapsed since v was generated and multicast. It follows that $v \in G_p^{(t)}$.

Extracting Every Honest Vertex Recall that an honest participant extracts the starting vertices in its graph and all their predecessors, and then outputs only the vertices which are far from the end of its graph. We show that an honest participant always extracts *every* honest vertex in its graph.

Lemma 3 (Extracting All Honest Vertices in a Local Graph). *For every time t, honest participant p active at t, and honest vertex $v \in \mathbb{G}^{(t)}$: $v \in G_p^{(t)} \implies v \in \mathsf{extract}(G_p^{(t)})$.*

The lemma follows by showing that every honest vertex v eventually gains at least one honest successor which is not too far from v, measured in terms of depth. Intuitively, after an honest vertex v is generated, the first vertex generated by an honest participant with v in its view must be a successor of v. It follows that for every honest vertex v which is not a starting vertex in an honest participant's graph, there must be a path from v to a starting vertex in the graph.

Consistency of Corrupt Vertices We show that consistency of corrupt vertices follows from consistency of their honest successors (or lack thereof). If every vertex is honestly generated and immediately multicast, then no vertex is ever orphaned. Only if a corrupt participant withholds a vertex can the vertex be orphaned. We show that after a corrupt vertex is generated, there is a limited time during which it must gain an honest successor or it will be orphaned. Imagine that starting at some time in an execution, corrupt participants use all of their resources to build a "withheld branch" B of $\mathbb{G}$ which includes no honest vertices, while honest participants continue to build $\mathbb{G}$ as per the protocol. Intuitively, if $\frac{\alpha}{\rho} > \beta$ (as we require), then the corrupt participants cannot keep pace with the honest participants, and eventually B will fall behind the depth of $\mathbb{G}$. We can compute for how long a withheld branch B can remain close in depth to $\mathbb{G}$. We derive a constant ℓ_2 for which any vertex which is ℓ_2 depth from the end of an honest participant's local graph and is a predecessor of a starting vertex must have an honest successor.

Lemma 4 (Honest Reachability Requirement for Extraction). *For all t, participant p active at t, and vertex $v \in \mathsf{extract}(G_p^{(t)})$: $\mathsf{D}(G_p^{(t)}) - \mathsf{D}(v) > \ell_2$ implies there exists an honest vertex u reachable from v such that $\mathsf{D}(u) - \mathsf{D}(v) \leq \ell_2$.*

Recall that an honest participant decides whether to output a vertex v only once v is $\ell^* = \ell_1 + \ell_2$ depth from the end of its local graph. If v is a predecessor of a starting vertex, then it must have an honest successor which is more than ℓ_1 depth from the end of the graph. This honest successor must be in every honest participant's local graph with depth sufficient to output v; therefore, because u must be extracted from every honest view in which it exists, every honest participant with local graph deep enough to output v must do so.

References

1. Joël Alwen and Björn Tackmann. Moderately hard functions: Definition, instantiations, and applications. In *TCC (1)*, volume 10677 of *Lecture Notes in Computer Science*, pages 493–526. Springer, 2017.
2. Adam Back et al. Hashcash-a denial of service counter-measure, 2002.
3. Christian Badertscher, Peter Gazi, Aggelos Kiayias, Alexander Russell, and Vassilis Zikas. Ouroboros genesis: Composable proof-of-stake blockchains with dynamic availability. In *ACM Conference on Computer and Communications Security*, pages 913–930. ACM, 2018.
4. Christian Badertscher, Ueli Maurer, Daniel Tschudi, and Vassilis Zikas. Bitcoin as a transaction ledger: A composable treatment. In *CRYPTO (1)*, volume 10401 of *Lecture Notes in Computer Science*, pages 324–356. Springer, 2017.
5. Shehar Bano, Alberto Sonnino, Mustafa Al-Bassam, Sarah Azouvi, Patrick McCorry, Sarah Meiklejohn, and George Danezis. Consensus in the age of blockchains. *CoRR*, abs/1711.03936, 2017.
6. Iddo Bentov, Pavel Hubácek, Tal Moran, and Asaf Nadler. Tortoise and hares consensus: the meshcash framework for incentive-compatible, scalable cryptocurrencies. *IACR Cryptology ePrint Archive*, 2017:300, 2017.
7. Iddo Bentov, Rafael Pass, and Elaine Shi. The sleepy model of consensus. *IACR Cryptology ePrint Archive*, 2016:918, 2016.
8. Iddo Bentov, Rafael Pass, and Elaine Shi. Snow white: Provably secure proofs of stake. *IACR Cryptology ePrint Archive*, 2016:919, 2016.
9. Bernardo David, Peter Gazi, Aggelos Kiayias, and Alexander Russell. Ouroboros praos: An adaptively-secure, semi-synchronous proof-of-stake protocol. Technical report, Cryptology ePrint Archive, Report 2017/573, 2017. http://eprint. iacr. org/2017/573, 2017.
10. Cynthia Dwork and Moni Naor. Pricing via processing or combatting junk mail. In *CRYPTO*, volume 740 of *Lecture Notes in Computer Science*, pages 139–147. Springer, 1992.
11. Lisa Eckey, Sebastian Faust, and Julian Loss. Efficient algorithms for broadcast and consensus based on proofs of work. *IACR Cryptology ePrint Archive*, 2017:915, 2017.
12. Juan Garay, Aggelos Kiayias, Rafail Ostrovsky, Giorgos Panagiotakos, and Vassilis Zikas. Resource-restricted cryptography: Revisiting mpc bounds in the proof-of-work era. Cryptology ePrint Archive, Report 2019/1264, 2019. https://eprint.iacr.org/2019/1264.

13. Juan A. Garay and Aggelos Kiayias. Sok: A consensus taxonomy in the blockchain era. *IACR Cryptology ePrint Archive*, 2018:754, 2018.
14. Juan A Garay, Aggelos Kiayias, and Nikos Leonardos. The bitcoin backbone protocol: Analysis and applications. In *EUROCRYPT (2)*, pages 281–310, 2015.
15. Juan A. Garay, Aggelos Kiayias, and Giorgos Panagiotakos. Consensus from signatures of work. Cryptology ePrint Archive, Report 2017/775, 2017. https://eprint.iacr.org/2017/775.
16. Yossi Gilad, Rotem Hemo, Silvio Micali, Georgios Vlachos, and Nickolai Zeldovich. Algorand: Scaling byzantine agreements for cryptocurrencies. Cryptology ePrint Archive, Report 2017/454, 2017. https://eprint.iacr.org/2017/454.
17. Lucianna Kiffer, Rajmohan Rajaraman, and abhi shelat. A better method to analyze blockchain consistency. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, CCS ?18, page 729?744, New York, NY, USA, 2018. Association for Computing Machinery.
18. Leslie Lamport, Robert Shostak, and Marshall Pease. The byzantine generals problem. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 4(3):382–401, 1982.
19. Andrew Lewis-Pye and Tim Roughgarden. A general framework for the security analysis of blockchain protocols. *CoRR*, abs/2009.09480, 2020.
20. Andrew Miller, Ahmed Kosba, Jonathan Katz, and Elaine Shi. Nonoutsourceable scratch-off puzzles to discourage bitcoin mining coalitions. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*, pages 680–691. ACM, 2015.
21. Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system, 2008.
22. Rafael Pass, Lior Seeman, and abhi shelat. Analysis of the blockchain protocol in asynchronous networks. *IACR Cryptology ePrint Archive*, 2016:454, 2016.
23. Rafael Pass and Elaine Shi. Rethinking large-scale consensus. In *Computer Security Foundations Symposium (CSF), 2017 IEEE 30th*, pages 115–129. IEEE, 2017.
24. Marshall Pease, Robert Shostak, and Leslie Lamport. Reaching agreement in the presence of faults. *Journal of the ACM (JACM)*, 27(2):228–234, 1980.
25. Benjamin Terner. Permissionless consensus in the resource model. Cryptology ePrint Archive, Report 2020/355, 2020. https://ia.cr/2020/355.
26. Florian Tschorsch and Bjrn Scheuermann. Bitcoin and beyond: A technical survey on decentralized digital currencies. Cryptology ePrint Archive, Report 2015/464, 2015. https://eprint.iacr.org/2015/464.