Permissionless Consensus in the Resource Model

Benjamin Terner

UC Santa Barbara bterner@cs.ucsb.edu

Abstract. Nakamoto's Bitcoin protocol inspired interest in the permissionless regime of distributed computing, in which participants may join and leave an internet-scale protocol execution at will, without needing to register with any authority. The permissionless regime poses challenges to the classical techniques used for consensus protocols, in which participants attempt to agree on a function of their inputs. Crucially, classical consensus techniques require honest participants to remain online and active, and to know an upperbound on the number of participants. Bitcoin addresses this issue by requiring Proof of Work in order to send a message in protocol, and other Bitcoin-inspired works have developed Proof of X variants to remediate the shortcomings of Proof of Work.

We propose an abstraction for Proof of X called *resources*, inspired by how many variants are used in practice. We then show that given few additional assumptions, resources are sufficient to achieve consensus in the permissionless regime. In particular, with appropriate assumptions about resources, it is not necessary to know a bound on the network delay, participants do not need clocks, and participants can join and leave the execution arbitrarily. The core idea is to shift focus from the proportion of honest parties in an execution to the proportion of messages sent by honest parties. We formally model consensus protocols in the permissionless regime, and show how to parameterize a permissionless execution using only the long-term proportion of resources acquired by honest participants and an upperbound on the rate at which resources enter the system, relative to the maximum network delay (without needing to know the network delay).

Along the way, we provide a generalized definition of blockchains which we call graph consensus. We present a protocol in the permissionless regime that achieves graph consensus, even when resources enter the system at high rates, but the required honest majority increases with the rate. We show how the protocol can be modified slightly to achieve one-bit consensus. Finally, we 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

1 Introduction

1.1 Permissionless Consensus and PoX

The distributed system problem of *consensus* has been studied for decades since the seminal works of [39, 29]. In the general form of a consensus protocol, 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. In the classical regime of consensus protocols, the participants are determined before a protocol execution begins. Crucially, most classical techniques for practical consensus protocols involve the use of quorum or threshold systems, for example [16, 9, 12, 28]. Famously, consensus on even a single bit in the classical regime is impossible unless more than two thirds of participants are honest [39, 29, 17, 9].

Recently, the advent of Bitcoin [35] ushered in renewed interest in consensus protocols by introducing the *permissionless* regime, in which a protocol execution is open to all those who want to participate. This 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, and the identities of the participants at any point in time are unknowable. Because

of dynamic participation, the permissionless regime presents challenges to the techniques usually employed by classical consensus protocols. Specifically, the permissionless regime lends itself to Sybil attacks, in which an adversary creates a large number of malicious identities and uses them to overwhelm the honest participants. This makes it challenging to implement quorums or depend on thresholds in the number of honest participants.

Proof of X Despite the challenges, consensus protocols for the permissionless regime have proliferated since Bitcoin [5, 22]. Bitcoin addressed the challenges of the permissionless regime by requiring participants to solve Proof of Work to participate [18, 2, 35]. In response to Proof of Work's wasteful computation, many Proof of X (PoX) variants have been proposed (see [5] for an overview), the most popular being Proof of Stake [8, 24, 3] (PoS). At a high level, most PoX systems require participants to attempt to solve some computational puzzle, and those who solve the puzzle are granted the right to send message containing the solution to all the other participants.

This technique serves multiple roles in taming the permissionless regime. First, the fact that participants can join and leave arbitrarily poses fatal issues for traditional protocols which may wait for a participant that has gone offline to send the next message. Because PoX systems require each participant to solve a puzzle in order to speak, they implement a self-selecting lottery to choose the next speaker from among those who are online. Second, Proof of X systems remediate the problem of Sybil attacks by inextricably tying participation in the protocol to *physical resources*. Although joining an execution and listening for messages may be free, sending messages is not. It is then possible to reason about security of a protocol based on whether an attacker does not control more physical resources than the honest participants. Recent work [36, 7, 14] even shows that they can maintain security when honest participants sustain a simple majority of the physical resources, which circumvents the classical requirement that more than two thirds of all participants are honest [39, 29, 17, 9].

Resources: A Unifying Abstraction It appears that PoX techniques tame the challenges of the permissionless regime. However, current permissionless consensus protocols do not use their chosen PoX as a black box, and their proofs cannot be separated from the assumptions on the PoX. Therefore, to understand the permissionless regime and the power of PoX, we ask:

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

In this work, we model a unifying abstraction of PoX which we simply call *resources*, an allusion to the fact that PoX systems tie participation to physical resources. To date, all other works we know about in the permissionless model also assume some synchronization assumption, either knowledge of the network delay [7, 8, 19] or (weakly synchronized) clocks [24, 14, 3, 4], plus some assumption about the number of active participants. But are these assumptions necessary given resources?

As a second goal, we introduce a framework for blockchain protocols which uses resources as a black box. If resources are implementable, then it should be possible to separate the proof of the blockchain protocol which uses resources from the implementation of resources. This improves modularity of proofs and allows new protocol designers to design new implementations of resources under reasonable assumptions.

Extending Classical Models with Resources By modeling resources, we can extend the vast classical consensus literature ([21, 16, 17, 1, 10] among many others) with PoX-style protocols. One can consider resources to be a bridge from the classical distributed systems literature to the field of resource-constrained cryptography.

1.2 Modeling Resources

Towards answering our fundamental questions, we model the properties of resources and show how we can parameterize an execution based on constraints over how participants acquire resources.

The Properties of Resources As we have explained, resources are often used in the permissionless regime to implement virtual lotteries based on physical resources. However, in practice, these lotteries have more useful properties than just selecting the next speaker. Consider that in most Proof of Work and Proof of Stake protocols [36, 7, 14, 24], participants attempt to solve their respective Proof-of-X puzzles by feeding a string to some cryptographic hash function, where the string is composed of a protocol message (most famously, a public key and a list of transactions) and a random nonce. A puzzle solution is found if the hash function output is less than some target value. The string fed to the hash function is both the witness to the solution *and* conveys the semantics of the protocol message.

In our model, rather than requiring participants to solve a PoX puzzle, we say that participants are *allocated* resources from the environment. We model the properties of resources by observing two qualities of the above process:

1. **Unforgeability** Resources are *unforgeable*, meaning no participant can fake the fact that it has a resource. This models the facts that Proof-of-X solutions are tied to physical resources and that Proof-of-X solutions can be verified by other participants.
2. **Binding** Participants must *bind* a string to their resources, where the string carries a protocol message. Importantly, the string must be chosen at the moment that the resource is generated, reflecting the fact that when attempting to solve a hash puzzle, the input must be chosen at the same time the hash function is invoked. Once a string is bound to a resource, it cannot be changed.

Parameterizing a Permissionless Execution In the permissionless regime that we define, we parameterize an execution using only two constraints on resources.

1. **Long Term Honest Majority** We let α denote the *long-term* proportion of honest resources. We say the *long-term* proportion because we also allow the corrupt participants to have a short-term advantage in resources. Specifically, in 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 ε represents a short-term corrupt advantage and $\beta = 1 - \alpha$ is the long-term proportion of corrupt resources. When $\alpha > \beta$, we say that honest participants receive a *long term* majority of resources.
   By introducing the parameter ε, we model the fact that an adversary may attempt to pool its physical resources in order to achieve a short "burst" of resources. However, if the corrupt participants perform a burst, then they must pay for it by depleting their capacity to receive more resources for some time after.

2. **Rate Limit** We let ρ upperbound the rate at which resources may be generated. Specifically, ρ is the maximum number of resources that may be generated per Δ time, where Δ is the unknown maximum network delay.

Remark 1 (On the Style of the Definitions). These definitions assume the style of traditional distributed systems literature, forsaking the standard UC framework [11]. Our approach is similar to a paradigm famously employed by coding theory. Consider that many encoding schemes offer guarantees of the style "as long as fewer than n bits of a codeword are mauled, the plaintext can be recovered." Our constraints to provide thresholds on adversarial behavior under which we prove security.

1.3 Main Results

One-Bit Consensus We find that resources *do* imply consensus with *very few* additional assumptions. We model a regime in which participants can join and leave an execution arbitrarily, they do not have clocks, they do not know the maximum network delay, and the number of participants which are online at any point of time is unknown. The we only assume knowledge of ρ, which upperbounds the rate at which resources are allocated relative to the (unknown) maximum network delay, and require that honest parties receive a majority of resources in the long term.

Theorem 1 (Informal). *For all $\alpha > \rho c(1-\alpha)$, where c is a derived constant on the order of $\rho + \varepsilon$, there exists a one-bit consensus protocol in the permissionless regime with resources.*

Graph Consensus The technique that we use to build one-bit consensus from resources is 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. Participants continuously output subgraphs of their local graphs. The security goals of a graph consensus protocol are generalizations of those proposed by [23, 36, 6]. Specifically, a graph consensus protocol should achieve two properties. First, *consistency* requires that for any two graphs output by honest participants, one participant's output must be a subgraph of the other. Second, *liveness* requires that honest participants may not trivially output empty graphs, but that their outputs must grow with the size of their local graphs.

Theorem 2 (Informal). *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 (recall that c is on the order of $\rho + \varepsilon$).

1.4 Technical Overview

The Permissionless Regime To show the power of resources, we model a regime in which traditional consensus techniques are inapplicable. In our regime, the maximum network delay Δ is unknown and participants do not have clocks. The number of participants is unknown to the participants *and may be unbounded*. The adversary can corrupt parties adaptively, and it controls resource allocations, subject to constraints on honest majority and the allocation rate.

Additionally, participants can join and leave the execution arbitrarily (participation is dynamic). In the extreme, this means that a protocol 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, this is just long enough to receive the state of the system and send a single message). In this regime, the state of the protocol must be maintained by the environment as it transmits the honest parties' messages, since honest parties are not active long enough for their own states to change.

One might think that achieving consensus in our regime is intractable; indeed, all classical techniques for consensus are inapplicable in our regime because we cannot synchronize rounds or threshold the proportion of honest participants. Moreover, Pass and Shi [7, 38] prove that in 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, this is because an adversary can always split the execution into two groups, and deliver messages within each group quickly but between groups slowly. If the adversary can deliver messages between groups more slowly than it takes each group to produce output individually, then no protocol can achieve consensus.

One might think to use resources in order to perform committee election, and then employ a classical consensus protocol among committee members (as in [24, 37, 26]). However, even agreeing on a committee is highly nontrivial without knowing Δ, and possibly implies consensus on its own if participants can output a function of the committee members' identities. Additionally, committee election requires that the committee members stay online long enough to participate in the secondary protocol, which is not guaranteed in our regime. We therefore require new techniques.

Building Consensus from Resources Despite the above difficulties, consensus is possible in our regime. Our protocols and their guarantees are independent of the number of honest participants and how long they stay online. We require only that honest participants *collectively* receive sufficiently more resources over the long term than the corrupt participants, as constrained by the parameters α and ε, and that the parties know a bound ρ on the rate at which resources are allocated. Without needing to know the communication delay Δ, the rate limit ρ acts as very weak connection between computing power and network synchronization. We remark on the strength of this connection in Section 1.5; although weaker than knowing Δ, this assumption suffices to subvert the impossibility of Pass and Shi.

Given a long-term majority of resources and a bound on the rate, honest participants can use the properties of resources to build a virtual, global directed acyclic graph (DAG) which captures the global (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 Proof of X). The unforgeability and binding properties of resources enforce that corrupt participants cannot manipulate the graph structure or create fake vertices. 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.

The technique requires that honest participants receive more resources over the long term than corrupt participants. In our graph protocol, we use the long term advantage similarly to many longest-chain blockchains. However, rather than measure the length of a chain in the DAG, we

define the depth of a DAG to be the length of the longest path from the root to a leaf 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 a predecessor 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.

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 In our model, we cannot use longest-chain or heaviest-chain protocols 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. Our definition of graph consistency (Definition 14) requires that if any two participants output graphs G_1 and G_2, respectively, at any points in time, then $G_1 \subseteq G_2$ or $G_2 \subseteq G_1$. In such an execution, no processor could ever output either branch of the fork. Our definition of liveness (Definition 15) lowerbounds the size of a graph output by an honest party by a function of the number of vertices in its local graph. In this execution, despite the fact that the honest parties' local graphs grow, liveness fails because they can never output any vertices. Note that this may happen even if the corrupt participants receive no resources. In a random model, forks are likely to be resolved eventually, which allows participants to eventually output one branch of the fork. The perpetual fork attack is also discussed in [25].

1.5 Comments about the Model

On Knowing the Rate Limit We believe it is reasonable to assume that a rate of resource allocations is known relative to the network delay *despite the fact that the network delay is unknown*. In practice, a Proof of Work system is parameterized by estimating the amount of time it takes to propagate a block through the network (as by [15]) and then tuning a hardness parameter to achieve a particular rate of puzzle solutions per conservative network delay time. Proof of Stake systems also tune their parameters to achieve a certain number of PoS solutions per round.

We remark that knowing a bound on the resource rate relative to the network delay is a weaker assumption than knowing the network delay. Given knowledge of the network delay, participants can execute synchronous protocols that proceed in rounds. This is possible because the adversary cannot manipulate honest participants' timekeeping abilities. However, a bound on just the resource rate does not directly yield synchronization, since the adversary may induce a large difference in two honest processors' views at the same moment in time by selectively delivering corrupt resources to one participant but not to another. Our protocols are roundless/continuous, and the participants operate *asynchronously* by responding to events in which they receive messages or resources.

Network with Multicast We assume a multicast functionality in which every honest message is delivered to all honest participants, but the upperbound Δ on the network delay is unknown. We imagine a network overlay that is sufficiently connected that between every two honest participants, there is at least one path through the network consisting of only honest participants. This guarantees that every message multicast by an honest node is eventually delivered to every other honest node. However, because the diameter of the network and the transmission delay between nodes is unknown, the maximum network delay Δ for a multicast is unknown. Corrupt participants can inject messages and selectively send messages to only some honest participants by corrupting the appropriate edge nodes.

We note that some form of multicast or broadcast functionality is necessary for any protocol in a permissionless regime. In [4], this is achieved through a network functionality that registers and unregisters participants as they enter and leave an execution and delivers messages to registered participants. Our approach resembles that of [36, 7], in which there exists known a network delay Δ, except that in our model, Δ is unknown to the participants.

The Need for a Recovery Protocol We assume that when a participant comes online, it immediately receives all messages which have been multicast or sent directly to it more than Δ time in the past. This assumption models the expectation that when participants come online, they execute some recovery protocol to receive the most recent state of the system (up to the communication delay) before participating in the protocol. An analogous recovery protocol in any blockchain requires a participant to download the blockchain up to the most recent blocks before it begins mining.

The need for such a recovery protocol is highlighted by [4], who showed that in Bitcoin, if participants can come online and generate blocks before they recover the state of the blockchain (up to messages pending over the network), it is impossible to prove anything. Intuitively, honest participants can be manipulated to generate blocks that are shallow relative to the length of the longest chain, which constitutes a (unwitting) deviation from the honest protocol.

Without modeling the many ways in which a participant may be online but desynchronized, we consider a participant to be honest at the moment it receives a resource if it is executing the honest protocol and it has received the state of the execution up to the messages which were sent within the last Δ time, and corrupt otherwise.

1.6 Implementing Resources

Our definitions are inspired by how we see resources used in practice. In Appendix A, we argue (without full proofs) that various forms of PoX implement resources. Future designs of PoX need only prove the properties we have delineated under reasonable assumptions for their applications, and then can plug into consensus protocols that exploit them.

1.7 Related Work

Comprehensive overviews of the blockchain literature can be found in the systemizations of knowledge by [22] and [5]. Here we describe only works we know about that solve similar problems or use similar techniques.

As far as we know, no other works present the common qualities of PoX via a single abstraction. However, Miller et al [32] model Proof of Work as scratch-off-puzzles, showing a number of desirable properties for Proof of Work objects.

Other works have studied one-bit consensus using proofs of work and blockchains. Among them, Miller and Laviola [33] show how to achieve anonymous consensus from moderately hard puzzles when the network delay is known. GKL [23] show how to achieve byzantine agreement in synchronous networks using the "Bitcoin backbone" protocol. EFL [19] construct broadcast and consensus from Proof of Work but require clocks and knowledge of the network delay.

A number of other works model permissionless blockchains with proof of work or proof of stake, most notably GKL [23], PSs [36] (followed by Pass and Shi [7]) and their respective successors. BMTZ [4] model the Bitcoin protocol in the UC model with dynamic player sets. Ouroboros Praos [14] models a Proof of Stake blockchain with semi-synchronous communication, and Ouroboros Genesis [3] presents their version of dynamic availability. The Ouroboros protocols (weakly) synchronize their participants via a global clock functionality. Among all the works studying PoX consensus protocols, we are the only one that we know in a deterministic model in which the adversary controls allocation.

There are many works that implement agreement on DAGs, specifically in attempts to scale the throughput of blockchain protocols "on the chain." The structure of the DAG built in our graph protocol bears some resemblance to SPECTRE [42] and PHANTOM [43], but our definitions, assumptions, and analyses are very different. The same is true for Meshcash [6], who adopt the model of [36]. The Avalanche protocol [40] also employs agreement on a DAG for high throughput of consensus instances for synchronous participants in a permissioned network.

2 Formal Model

We denote by $\mathbb{N}$ the natural numbers and by $\mathbb{R}_{\geq 0}$ the set of non-negative real numbers. Fix an alphabet Σ. Let $\mathcal{M} = \Sigma^*$ be the set of strings (we also call them messages) over the alphabet. We let $\mathcal{ID}$ denote the set of identities, and let $\mathsf{broadcast}$ be a special symbol not in $\mathcal{ID}$. We denote by Ψ the set of resources.

We use $||$ to denote concatenation. We denote by ϵ the empty string. For a set S, we use the notation $S_{\varnothing}$ to denote $S \cup \{\varnothing\}$. We let $\mathcal{P}(S)$ denote the powerset of S. We let $|S|$ denote the cardinality of S.

2.1 Model of Computation

Automata In this work, we model computation via automata that can send and receive messages. We begin by introducing an interactive automaton:

Definition 1 (Interactive Automaton). *An interactive automaton is a tuple*

$$(\mathcal{Z}, \mathcal{ID}, \Sigma, \mathcal{Z}_{\mathsf{init}}, \delta, \lambda, \xi)$$

Let $\mathcal{M} = \Sigma^*$ and $\Theta = \mathcal{ID}_{\varnothing} \times \mathcal{M}$. The elements of the above tuple are: $\mathcal{Z}$ is an infinite set of states; $\mathcal{ID}$ is a set of identities; Σ is an alphabet; $\mathcal{Z}_{\mathsf{init}} \subseteq \mathcal{Z}$ is a set of initial states; δ is a state transition function

$$\delta : \mathcal{Z} \times \Theta \mapsto \mathcal{Z}$$

; λ is a communication function

$$\lambda : \mathcal{Z} \times \Theta \mapsto \mathcal{P}((\mathcal{ID} \cup \{\mathsf{broadcast}\}) \times \mathcal{M})$$

; and ξ is an output function

$$\xi : \mathcal{Z} \mapsto \mathcal{M}$$

We intuitively understand the execution of interactive automata as follows. An automaton begins its execution in some starting state $z_{\mathsf{init}} \in \mathcal{Z}_{\mathsf{init}}$, and its state transitions and outputs are determined by the inputs provided to its functions δ,λ, and ξ. For example, let $z \in \mathcal{Z}$ be the current state in some transition. On input $\theta \in \Theta$, the automaton computes $\nu \leftarrow \lambda(z, \theta)$ as the set of messages to send to other automata, and it transitions to the next state $z' \leftarrow \delta(z, \theta)$. When the automaton transitions, it also updates its output to $\xi(z')$.

By modeling interactive automata, we intend to model a group of automata that interact by sending and receiving messages to each other. When an automaton A's communication function λ outputs (id, m), we can understand that A sends m to an automaton with identity id.

Protocol A *protocol* describes how a group of interactive automata interact with each other. Concretely, a protocol Π is a triple $(\mathcal{ID}, \Sigma, \Gamma)$ where $\mathcal{ID}$ is a set of identities, Σ is an alphabet, and Γ is a mapping from identities to interactive automata. For a given $\mathsf{id} \in \mathcal{ID}$, we let A_{id} denote the automaton specified by Π's mapping Γ, where $\mathsf{A}_{\mathsf{id}} = (\mathcal{Z}_{\mathsf{id}}, \mathcal{ID}, \Sigma, \mathcal{Z}_{\mathsf{init,id}}, \delta_{\mathsf{id}}, \lambda_{\mathsf{id}}, \xi_{\mathsf{id}})$.

All of the protocols we present are parameterized by an arbitrary, unbounded set $\mathcal{ID}$, which for convenience we think of as $\mathbb{N}$. In addition, for all of the protocols we consider, $\Sigma \supseteq \{0, 1\} \cup \Psi$, where Ψ is the set of resources. In the discussions of our protocols, we omit $\mathcal{ID}$ and Σ from the notation and just give the mapping Γ. Because in our protocols, Γ gives the same automaton for every identity, when we describe a protocol Π, we will just describe the automaton to which all identities are mapped.

For convenience, when discussing an execution of a protocol, we refer to an interactive automaton which sends or receives messages in an execution as a *participant*.

2.2 Our Approach

In this work, we model consensus protocols with unboundedly many participants, a setting inspired by blockchains. We now highlight a few key aspects of our approach which differ from classical consensus works.

Number of Participants In classical consensus, the number of participants in a protocol would be a parameter of the protocol. We consider protocols in which the number of participants may be unbounded. This has been considered in recent blockchain literature [4, 3].

Time We adopt an abstract notion of global time to describe a protocol's execution. Time is a totally ordered set of points, $\mathcal{T} = (\{t_1, t_2, t_3, \ldots\}, <)$. We will use addition and subtraction operations defined over the set of points to discuss the elapse of time. Without loss of generality, in this work we think of $\mathcal{T} = \mathbb{N}$ and we let 0 be the starting time of any execution. We emphasize that automata in our model *do not have clocks* and cannot measure the global time.

Resources We introduce special black-box objects called *resources*. At first glance, one can conceptualize resources as an abstraction of proof of work. In our model they are unforgeable objects created by an external entity, and are allocated to participants by the entity. Lucky participants who receive resources gain extra power in protocols that exploit them.

Resources alone are not useful because they do not convey semantics, so to enable protocols to use them in meaningful ways, we let resources be *bound* to messages. One can think of binding a resource to a message as a way for a protocol to assign special elite status to the message.

Fix a set of resources Ψ, and let $\mathcal{M} = \Sigma^*$ be the set of strings over an alphabet Σ. The set of *bound resources* $\Psi^{\mathcal{M}}$ is $\Psi \times \mathcal{M}$. For some $\psi \in \Psi$ and $m \in \mathcal{M}$, we denote by ψ_m the corresponding element of $\Psi^{\mathcal{M}}$. The bound resource ψ_m is encoded as $\psi||m||\psi$.

In this work, a resource can be bound only once. After being bound to a message, a resource cannot be bound to another message. One can think of this property as similar to proof of work in Bitcoin [35], in which a proof of work is associated with a single string, but finding another string that yields the same proof of work is computationally hard.

2.3 Execution of a Protocol

Environment Many nondeterminstic factors influence a real-world protocol execution. For example, participants' hardware do not all run at the same speed, and messages sent over a network are not delivered immediately. We model the nondeterminism of an execution using an *environment*. An environment $\mathcal{E}$ is an (non-interactive) infinite automaton that directs an execution of a protocol by:

- scheduling when participants transition between states and send messages,
- scheduling message deliveries,
- and importantly, allocating resources to participants.

Buffers Participants in an execution communicate by sending messages to each other. Each participant in an execution has a buffer, which is an unordered multiset containing elements of $\Theta = \mathcal{ID}_{\varnothing} \times \mathcal{M}$, that stores messages sent by other participants. We denote by $\mathsf{buff}_{\mathsf{id}}$ the buffer of p_{id}. As we will explain in detail, when a participant sends a message, the environment immediately delivers the message to its recipient's buffer. The environment then chooses when to deliver messages from the buffer to the recipient.

Configuration and Events To fully describe how the environment directs an execution, we explain how the state of a protocol execution evolves over time. We use two primitives: configurations and events.

1. A *configuration* of a protocol execution is a snapshot of the set of states of all participants and their buffers at a point in time.
2. An *event* is a single state transition by a single participant. We call the transitioning participant the *agent* of the event, and we also say that the agent is *active* during an event.

During an event, the agent receives messages that have been sent to it. In addition, the environment can optionally *allocate a resource* to the agent, which is the process by which participants acquire new resources. The environment directs an execution by *applying* events to the execution's configuration, defined as follows: when $\mathcal{E}$ applies event e with agent p in state z to configuration C at time t,

– **State Transition:**
  - If p has not been the agent of an event at any $t' < t$, $\mathcal{E}$ selects p's initial state z.
  - $\mathcal{E}$ removes a (possibly empty) subset of buff_p, which we call μ, and delivers all of μ to p. We say the elements of μ are *received* by p at t. $\mathcal{E}$ also optionally chooses a resource ψ to allocate to p.
    $\mathcal{E}$ chooses the order in which to deliver each element of μ to p. (Notice that this helps capture that the order of message deliveries may be arbitrary.) It begins by removing some element θ from μ, and invoking both δ and λ on (z, θ). For each successive $\theta' \in \mu$, it invokes δ and λ on (z', θ'), where z' is the output of the previous invocation of δ and θ' is the next element of μ selected by $\mathcal{E}$. At the end, the environment collects all the outputs of λ into a single set ν.
    If $\mathcal{E}$ selects an empty subset of buff_p, it invokes p's transition and communication functions once on $(\varnothing, \epsilon)$.
  - If p is allocated a resource ψ, $\mathcal{E}$ invokes p's transition function and communication function on $(\varnothing, \psi)$. In any event in which p is allocated a resource, the environment invokes p's state transition function on the resource last. If p is allocated a resource ψ, p immediately pairs ψ with some $m \in \mathcal{M}$ to obtain a bound resource ψ_m. (The choice of ψ_m is reflected in the state to which p transitions.)

– **Message Delivery:**
  - Let ν be the set of pairs output by the agent's communication function. For each pair $(\mathsf{broadcast}, m) \in \nu$ the environment immediately delivers m to the buffer of every participant, and for each pair $(\mathsf{id}, m) \in \nu$ it delivers m to $\mathsf{buff}_{p_{\mathsf{id}}}$.
    We model authenticated communication by saying that when $\mathcal{E}$ delivers m to its recipient, it does so by encoding the pair (id', m), where id' is the identity of the sender, into the recipient's buffer. (We note that our protocols do not use authenticated communication.)

Simultaneous Events The environment directs the execution of a protocol by keeping track of the execution's configuration and, for every $t \in \mathcal{T}$, choosing a (possibly empty) set of events to apply to the configuration, subject to the constraint that all events at any time t must have distinct agents. We call a *step* τ a set of events chosen by the environment to apply at a single t. Note that at any time t, $\mathcal{E}$ may select no events to apply to the configuration.

When $\mathcal{E}$ chooses multiple events for some step τ, it applies the events within τ in parallel. First it invokes the state transitions of the agents of all events, and then it delivers all messages specified by the participants' communication functions. In particular, this means that if p and q are both agents of events at t, no message sent by p at t can be received by q at t, and vice versa.

Corruption The above describes an honest execution of a protocol. However, the environment has the ability to corrupt participants arbitrarily. To corrupt a participant, the environment replaces the participant's transition and communication functions with functions of its choice. Once corrupted, a participant stays corrupted and may not be corrupted again. We model corruptions this way in order to capture identities implemented using key pairs. In a typical real-world scenario, if a participant's key pair is compromised (or lost), it must assume a new identity.

2.4 Transcript

An execution of a protocol produces a transcript σ that records every event and corruption in the execution. Events are written as tuples containing a timestamp t, the identity id of the agent, and the inputs to the state transition function:

1. **Event** is $(t, \mathsf{id}, z, \mu, \psi)$, where p_{id} begins its state transitions in state z at t, μ is the finite, possibly empty set of pairs in Θ delivered to p_{id}, and $\psi \in \Psi$ is allocated to p_{id} (if no resource is allocated, this element is $\varnothing$).

Corruptions appear the transcript as tuples containing the new transition function δ and communication function λ that the agent begins to follow:

2. **Corruption** is $(t, \mathsf{id}, \delta, \lambda)$, where δ and λ are the transition and communication function executed by p_{id} starting at t.

The contents of a transcript are partially ordered by their timestamps. (Notice that for most transcripts there exist many equivalent transcripts in which events are reordered; to avoid a lengthy treatment, we defer to Lamport [27].)

We also define the view of a participant at some point in time. The *view* of a participant p at some time t in an execution is the set of events for which p is the agent from the beginning of the execution until t. In particular, we will say that a resource ψ is in a participant p's view at time t if it has appeared in an event for which p is the agent (either as part of a message or as an allocated resource) before or at t.

Notice that a transcript is a complete record of a protocol execution. From it one can infer the configuration at every time in the execution.

Remark 2. Although we could model our automata and environment using Turing machines in the UC model, we choose to model our executions using automata. We view our work as an extension of the classical distributed systems literature that defined consensus and modeled their executions by applying events to a set of automata in some configuration. All of our ideas can be adapted to the UC model. Finally, the above model does not consider efficiency. Although it appears our automata are able to achieve very expensive computations in a single state transition, the computations performed in the protocols we present are efficient.

2.5 Constraints

We now define constraints on an environment that take the form of assumptions when designing a protocol. We introduce a bit of notation to facilitate our definitions. In any transcript σ, let $\sigma^{(t,t')}$ be the subsequence of σ containing all of the events that happen between times t and t', inclusive of t and t'.

Synchronization In our setting, participants operate asynchronously and communication is partially synchronous. We adapt the definition of asynchronous computation from [21] and [16], and we adapt partially synchronous communication from [17].

Definition 2 (Φ-Synchronous Computation). *For a constant Φ, participants in an execution described by transcript σ are Φ-synchronous if for every time t and every $t' > t$, if any participant is the agent of Φ events in $\sigma^{(t,t')}$, then every other participant is the agent of at least one event in $\sigma^{(t,t')}$.*

Participants in an execution are *asynchronous* if there does not exist a Φ that constrains σ. We model participants as asynchronous in order to capture the membership of permissionless internet-scale protocols, which often exhibit heterogenous hardware joining, leaving, and returning to an execution.

Definition 3 (Δ-Synchronous Communication). *For a constant Δ, communication in an execution defined by transcript σ is Δ-synchronous if for any message m that is placed in any participant p's buffer at time t, if p does not receive m before $t + \Delta$, then p receives m at its first activation at or after $t + \Delta$.*

When computation is asynchronous but communication is synchronous, participants can go through long periods of no activation, then "wake up" and receive many messages. This property has recently been framed in the "sleepy" model [7], but had been analyzed as early as [1] and [16].

A crucial difference between our model and previous works with asynchronous computation is that in our model, participants do not "know" the communication delay, meaning it is not given as input and not hard-coded into their states and transition functions. This notion was originally introduced by [17], who called the assumption *partially synchronous communication.*

Resources We introduce constraints on an execution regarding resources.

Definition 4 (Unique Resource Encoding). *An execution described by transcript σ satisfies* unique resource encoding *if*

1. *every bound resource ψ_m is properly encoded as $\psi||m||\psi$, and*
2. *for any two bound resource encodings $\psi||m||\psi$ and $\psi'||m'||\psi'$, if $\psi = \psi'$ then $m = m'$.*

A resource can be bound only once. After being bound to a message, a resource cannot be bound to another message. This constraint is inspired by the fact that PoX puzzle solutions are associated with a single string (the hash input), and a solution cannot be associated with any other strings.

Definition 5 (Respecting Resource Allocation). *We say a transcript σ* respects resource allocation *if no resource appears in σ before it is allocated.*

This definition seeks to capture the property that resources cannot be "made up" by participants. Just as no participant can solve the a PoX puzzle without evaluating some function, no participant should be able to send a bound resource to another participant if the resource has not been allocated.

Admissible Execution We define an *admissible* execution as one that satisfies the previous two constraints.

Definition 6 (Admissible Execution). *An execution described by transcript σ is* admissible *if it respects resource allocation and obeys unique resource encoding.*

In this work, we consider exclusively admissible executions.

Resource Allocations and Corruption Resource allocations are constrained both by the rate of allocation and by the proportion of honest allocations.

Definition 7 (ρ-Rate-Limiting). *Let $\rho \in \mathbb{N}$ and let Δ be the communication synchronization constant, or the network delay. An execution described by transcript σ with network delay Δ is* ρ-rate-limited *if for all t, there are at most ρ resources allocations in $\sigma^{(t,t+\Delta)}$.*

Although we assume that Δ is unknown to the participants, we assume that ρ is known. This combination of assumptions is inspired by the Bitcoin network, in which the diameter of the network is unknown but the rate of puzzle solutions (which implement resources) is known.

Every execution is affected by how many resources are allocated to honest participants and how many resources are allocated to corrupt participants. We introduce the following notation to denote how many resources are allocated to honest and corrupt participants over some span of time.

Definition 8 ($\Psi_{\sigma}^{(t,t')}$,$\Psi_{\mathsf{hon},\sigma}^{(t,t')}$,$\Psi_{\mathsf{cor},\sigma}^{(t,t')}$). *Let σ be the transcript of an execution. We denote the sets of resources allocated by the environment to all participants, honest participants, and corrupt participants between times t and t' in σ as follows:*

- $\Psi_{\sigma}^{(t,t')}$ *is the set of all resources allocated in* $\sigma^{(t,t')}$.
- $\Psi_{\mathsf{hon},\sigma}^{(t,t')}$ *is the set of resources allocated to honest participants in* $\sigma^{(t,t')}$.
- $\Psi_{\mathsf{cor},\sigma}^{(t,t')}$ *is the set of resources allocated to corrupt participants in* $\sigma^{(t,t')}$.

Because the execution is always implied or clear from context when we use this notation, we suppress the subscript σ from these variables.

The ratios of resources allocated to honest participants and corrupt participants are important parameters in an execution. We denote them as follows:

Definition 9 (α,ε-honest resource allocation, β,ε-corrupt resource allocation). *Let $\alpha \in [0,1]$ and let $\varepsilon \in \mathbb{N}$. An execution satisfies α,ε-honest resource allocation if for all times $t,t' > t$: $|\Psi_{\mathsf{hon}}^{(t,t')}| \geq \alpha|\Psi^{(t,t')}| - \varepsilon$. Equivalently, let $\beta = 1-\alpha$. An execution satisfies β,ε-corrupt resource allocation if for all times $t,t' > t$: $|\Psi_{\mathsf{cor}}^{(t,t')}| \leq \beta|\Psi^{(t,t')}| + \varepsilon$.*

Intuitively, α and β capture the long-term ratios of honest and corrupt resource allocations, respectively, and ε represents a small amount of "slack" in the ratios. ε also captures the short term advantage that corrupt participants may obtain in receiving resources.

3 One-Bit Consensus Problem

Our one-bit consensus problem is very similar to classical consensus [16, 17]. Every participant has a bit $b \in \{0,1\}$ as input, and they all attempt to *decide* on the same output bit. A participant *decides* by choosing an output bit which it may never change. At every moment that a participant is active, it produces an output in $\{0,1,\perp\}$, where $\perp$ means "undecided." For a participant p active at time t, we let $\mathsf{out}_p^{(t)} \in \{0,1,\perp\}$, denote its output. If for participant p active at time t, $\mathsf{out}_p^{(t)} \in \{0,1\}$, then for all $t' > t$ at which p is active, $\mathsf{out}_p^{(t')} = \mathsf{out}_p^{(t)}$.

Properties of an Execution As in the classical problem, the goal of a one-bit consensus protocol is to satisfy *agreement*, *nontriviality*, and *termination*. We say a protocol Π satisfies property P if every execution of Π satisfies P.

Definition 10 (Agreement). *An execution satisfies* agreement *if for all times t,t' and honest participants p,q active at t and t', respectively, $\mathsf{out}_p^{(t)} \neq \mathsf{out}_q^{(t')} \implies \perp \in \{\mathsf{out}_p^{(t)}, \mathsf{out}_q^{(t')}\}$.*

Definition 11 (Nontriviality). *A execution satisfies* nontriviality *if when all honest participants have the same input b, then for every time t and honest participant p active at t, $\mathsf{out}_p^{(t)} \neq \perp \implies \mathsf{out}_p^{(t)} = b$.*

Our definition of termination differs slightly from classical definitions. Recall that a resource is in a participant's view if it is allocated to the participant or if it appears in a message that the participant receives. We require participants to terminate only if sufficiently many resources have entered their views. In comparison, classical definitions require participants to terminate after finitely many steps.

Definition 12 (Termination). *An execution satisfies* termination *if there exists a positive integer R^* such that for every participant p active at time t with at least R^* resources its view, $\mathsf{out}_p^{(t)} \neq \perp$.*

4 Graph Consensus Problem

4.1 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, as follows:

Definition 13 (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 *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.

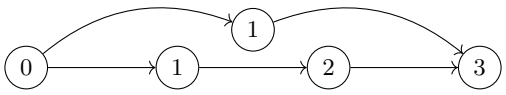

Fig. 1. An example graph in which each vertex is labeled with its depth. The root vertex has depth 0 by definition, and every other vertex's depth is defined by the longest path from the root to the vertex.

4.2 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 14 (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 size of its local graph G_p, as follows:

Definition 15 (f-Liveness). *Let $f\colon \mathbb{N} \times (0,1] \times \mathbb{R}_{\geq 0} \times \mathbb{N} \mapsto \mathbb{N}$. An execution satisfies* f-liveness *if for every time t and honest participant p active at t:*

$$|G_p^{*(t)}.V| \geq f(|G_p^{(t)}.V|, \alpha, \varepsilon, \rho)$$

and where α, ε, and ρ are parameters of the execution.

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-liveness to quantify the guaranteed proportion of honest vertices in a participant's output graph. When presenting f-liveness and h-liveness together, we use the term f, h-liveness.

Definition 16 (h-Liveness). *Let $h\colon \mathbb{N} \times (0,1] \times \mathbb{R}_{\geq 0} \times \mathbb{N} \mapsto \mathbb{N}$. An execution satisfies* h-liveness *if for every time t and honest participant p active at t:*

$$|\mathsf{hon}(G_p^{*(t)}.V)| \geq h(|G_p^{(t)}.V|, \alpha, \varepsilon, \rho)$$

where α, ε, and ρ are parameters of the execution.

5 Main Protocol

5.1 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 (Def 9), and the maximum rate of resource allocation ρ (Def 7).

Each participant p maintains a local directed acyclic graph (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 receives

messages. Whenever p is allocated a resource, it adds the resource to its graph 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. The keys are to show how a participant p chooses the predecessors of each vertex that it adds to its graph, and how each participant p computes its output G_p^* from its local graph G_p.

We describe resources as vertices as follows. When an honest 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) are still gaining successors.

Encoding a Graph Using Resources Recall that in our model, a resource is a black box object which is bound to a string that conveys its semantics. When a resource is allocated, the string is bound to the resource immediately and cannot be changed afterwards (by unique encoding, Definition 4). 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 message that is not the encoding of a graph 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 $G'.V$.

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, a participant 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 computes $G_p^* \leftarrow \mathsf{extract}(G_p)|_{\mathsf{D}(G_p)-\ell^*}$, meaning p outputs the vertices in its extracted subgraph with depth less than $\mathsf{D}(G_p) - \ell^*$, where ℓ^* is derived from the protocol parameters.

Remark 3 (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.

Properties of an Execution We now observe a number of useful properties of an execution which are inherited from resources.

Constrained Vertex Generation A participant can generate a vertex *only* when it is allocated a resource. It immediately follows that the set of all vertices that have been generated in an execution at some point in time is the set of resources that have been allocated in the execution up to that point in time. Furthermore, constraints on the rate at which vertices are generated and the proportion of honest vertices in an execution inherit directly from the respective constraints on resource allocation. Specifically,

- the rate at which vertices are generated in an execution is also upperbounded by ρ vertices per Δ time (Definition 7), and
- the proportion of vertices generated by honest participants is the proportion of honest resources allocated in an α, ε-honest execution (Definition 9)

Consistency Properties of Local Graphs We say that a graph G is *completely described* if for every vertex v in G, every one of v's predecessors is also in G. The protocol specification enforces the invariant that every honest participant's local graph is always completely described. Recall that each participant's graph is initialized to a graph with only the root vertex, and that participants' local graphs grow when they generate vertices and when they receive messages. No participant's local graph can become incompletely described when it generates a vertex, and any message that might cause a graph to be incompletely described is discarded. Therefore, if v is in an honest participant's local graph, then all of v's predecessors must be in the graph as well.

Recall that each vertex that is generated is uniquely committed to its predecessors. Because of this and the fact that every honest participant's local graph is always completely described, it follows that if v is in both $G_p^{(t)}$ and $G_q^{(t')}$, then v's predecessor graph is the same in both graphs. Moreover, it is immediate that $\mathsf{D}_{G_p^{(t)}}(v) = \mathsf{D}_{G_q^{(t')}}(v)$.

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 ψ
   - $G_p \leftarrow \mathsf{addVert}(G_p, \psi)$
   - multicast G_p
   - $G_p^* \leftarrow \mathsf{extract}(G_p)|_{\mathsf{D}(G_i)-\ell^*}$

Internal Functions:

1. $\mathsf{addVert}(G, \psi)$:
   - $V' \leftarrow \{u \in G.V \colon \mathsf{D}(G) - \mathsf{D}(u) < c \text{ and } \mathsf{outdegree}(u) = 0\}$
   - 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. $\mathsf{extract}(G)$:
   - $S \leftarrow \{v \in G.V \colon \mathsf{D}(G) - \mathsf{D}(v) \le c + \rho\}$
   - 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

Temporal Ordering Consider that because participants cannot "make up" resources (Definition 5), at the moment when the inbound edges for a vertex v are chosen, v cannot have an inbound edge from any vertex u which has not yet been generated. Because each vertex is uniquely committed to its predecessors, it follows that the predecessor-successor relations among vertices in a participant's local graph obey the temporal order in which the vertices are generated. Specifically, for all vertices v and u in any participant's graph, if v is generated before u in the execution, then u cannot be a predecessor of v.

5.2 Theorem Statement

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

Theorem 3. *For 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 $f(N, \alpha, \varepsilon, \rho) = h(N, \alpha, \varepsilon, \rho) = \alpha N - \varepsilon - \rho(\ell^* + 1)$.*

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.

We present the following proposition, which is the main desideratum of the proof of graph consistency.

Proposition 1. *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.

It is easy to see that 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).

5.3 Proof Overview

We now overview the proof of Proposition 1. The full proofs of Proposition 1 and Theorem 3 are in Appendix B.

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 p has been "orphaned." We call a vertex "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 *all* honest vertices are eventually output by honest participants, and intuitively, that no honest vertex is ever orphaned. 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 $v \in \mathbb{G}^{(t)}$:*

$$\mathsf{D}(G_p^{(t)}) - \mathsf{D}(v) > \ell_1 \implies v \in \mathsf{extract}(G_p^{(t)})$$

To prove this lemma, we first show 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. Second we show 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 (Definition 7) 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 in order to compute its output, an honest participant extracts the starting vertices in its graph and all their predecessors, and then outputs only those which are far from the end of its graph. We show that, in fact, 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. We use this to show 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. It also follows that no honest vertex is ever orphaned.

Lemma 1, consistency of honest vertices in participants' outputs, follows trivially from composition of Lemmas 2 and 3.

Consistency of Corrupt Vertices 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. Moreover, because every honest vertex is guaranteed to indefinitely gain honest successors, a corrupt vertex with an honest successor is guaranteed the same. Therefore, only a corrupt vertex with no honest successor in $\mathbb{G}^{(t)}$ can ever be orphaned. We complete the proof by showing that any corrupt vertex output by an honest participant p *must* have an honest successor in p's graph. Consistency of corrupt vertices follows from consistency of their honest successors (or lack thereof).

Withholding Vertices 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 $\alpha \approx \beta$, then B can grow at the same pace as $\mathbb{G}$ or even grow to be deeper than the rest of $\mathbb{G}$. However, if $\alpha > \beta\rho$ (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 $\mathsf{D}(u) - \mathsf{D}(v) \leq \ell_2$.*

Consistency of Corrupt Vertices Via Honest Successors 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.

6 From Graph Consensus To One-Bit Consensus

6.1 A Generic Transformation

We now show that one-bit consensus is implied by any graph consensus protocol which guarantees a long-term majority of honest vertices are accepted by honest processors. Specifically, we show that for any protocol that satisfies (a) graph consistency and (b) h-liveness such that there exists some N^* for which for all $N \geq N^*$: $h(N, \alpha, \varepsilon, \rho) > \frac{N}{2}$, there must exist a one-bit consensus protocol secure under the same parameters.

Theorem 4. *For any graph consensus protocol* Π *that satisfies both graph consistency and* h*-liveness for which there exists some* N^* *for which for all* $N \geq N^*$*:* $h(N, \alpha, \varepsilon, \rho) > \frac{N}{2}$*, there exists a one-bit consensus protocol that satisfies agreement, termination, and nontriviality under the same parameters.*

Proof. The proof transforms Π into a one-bit consensus protocol. We let Π^b represent the transformed protocol. The transformation works as follows. Whenever a participant generates a vertex, it binds an additional one-bit label, which is the participant's input bit, to the vertex. The participants run Π^b without producing output until a majority of the vertices output by the underlying Π must be honest vertices, and then they compute a majority of the bit labels of the graph output by Π. Termination follows because any honest participant with enough vertices in its graph can output a bit. Nontriviality follows because a majority of the parties' extracted vertices must be honest vertices. Agreement follows because honest participants compute the majority bit of vertex labels in *the same* output graph.

Honest participants run Π^b until they can output from their local graphs the smallest graph containing at least $\frac{N^*}{2}$ vertices. By h-liveness, there must be some point at which honest participants can output a graph with at least $\frac{N^*}{2}$ vertices. If not, then there would not be there exists an N^* such that for any honest participant p's local graph $G_p^{(t)}$ at time t for which $|G_p^{(t)}.V| > N^*$, that $|\mathsf{hon}(G_p^{*(t)}.V)| \geq \frac{|G_p^{(t)}.V|}{2} \geq \frac{N^*}{2}$.

We must argue that honest participants identify the *same* smallest graph containing at least $\frac{N^*}{2}$ vertices. We argue that in every execution, each honest participant's output graph must be partially ordered, and that any two participants' graphs must obey the same partial ordering. Assume that in some execution there is no such a partial ordering of vertices of honest participants' output graphs. Then it may be the case that for two honest participants p and q active at t and t', it is possible that that $G_p^{*(t)} \not\subseteq G_q^{*(t')}$ and that $G_q^{*(t')} \not\subseteq G_p^{*(t)}$. But this is a contradiction with the fact that Π satisfies graph consistency. However, it may be the case that some vertices in the participants' output graphs cannot be ordered relative to each other, (i.e. there are vertices u,v such that $u \not\prec v$ and $v \not\prec u$) so there may not be an output graph containing exactly $\frac{N^*}{2}$ vertices. Therefore, honest

participants identify the smallest graph containing at least $\frac{N^*}{2}$ vertices by the partial ordering of their outputs.

6.2 Our One-Bit Consensus Protocol

We now show how to achieve one-bit consensus by slightly modifying Π^G. Our protocol Π^{bit} differs slightly from the generic transformation provided in Section 6.1 for simplicity of presentation and proof.

We modify the graph consensus protocol as follows. Whenever a participant generates a vertex, it binds an additional one-bit label, which is simply the participant's input bit, to the vertex along with the vertex's edges. The participants run Π^G without producing output until their local graphs reach depth $k^* + \ell^*$, where ℓ^* is the same as in Π^G and k^* is an additional constant derived from the protocol parameters. For any participant p active at time t for which $\mathsf{D}(G_p^{(t)}) \geq k^* + \ell^*$, the participant outputs $\mathsf{extract}(G_p^{(t)})|_{k^*}$ from the graph consensus subprotocol. As its one-bit consensus output, p computes the one-bit label that is bound to a majority of extracted vertices. Even after a participant produces its output bit, it must continue to participate in the underlying execution of Π^G indefinitely; we explain why in a remark below.

Figure 3 describes Π^{bit}, our protocol for one-bit consensus. Π^{bit} is parameterized by α, ε, and ρ, which describe the ratio of honest resources and the maximum rate of resource allocation.

Protocol 2 DAG Protocol for One-Bit Consensus $\Pi^{\mathsf{bit}}(\alpha, \varepsilon, \rho)$

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. $x = c\varepsilon + c + \rho + \frac{\varepsilon}{\rho} + 1$
5. $\omega = \frac{\beta\rho}{\alpha}(x + \gamma + \frac{\varepsilon}{\rho} + 1) + \varepsilon$
6. $k^* = \frac{\omega + 2\varepsilon}{\alpha - \beta}$

Input

1. Each participant has a 1-bit input b

Internal Variable

1. $G_p = (V_p, E_p)$ is a participant's local state. Initially, $G_p = (\{\mathsf{root}\}, \emptyset)$

Protocol

1. **Framework** Run Protocol Π^G
2. **Labeling Vertices** Whenever a participant is allocated a resource, it additionally binds a one-bit label to the vertex it generates, where the label is the participant's input b
3. **Output** If $\mathsf{D}(G_p) > k^* + \ell^*$, output the majority bit in the labels of all vertices in $\mathsf{extract}(G_p)|_{k^*}$. Ties are broken by outputting 1.

Fig. 3. Protocol for one-bit consensus using graph consensus

Remark 4 (Indefinite Execution). Note that although honest participants may produce their outputs when their local graphs reach a fixed depth, it is important that honest participants continue to run the underlying graph consensus protocol indefinitely, until the execution ends. The reason

is straightforward: if ever honest participants stop running the underlying graph protocol, then corrupt participants can, with enough time, run an execution on their own which builds a deeper graph, with the property that the labels bound to vertices in the second graph would induce a decision of the opposite bit. This could cause disagreement with any honest participant that "wakes up" long after honest participants stop building the original DAG, and is presented with the two competing graphs.

Theorem 5. *For all ρ and all ε, and for all $\alpha > \rho(1-\alpha)((3-\alpha)\rho + \frac{\varepsilon}{\alpha} + \frac{\varepsilon}{\rho} + \varepsilon + 1)$ every every (α, ε)-honest, ρ-rate-limited admissible execution of $\Pi^{\mathsf{bit}}(\alpha, \varepsilon, \rho)$ satisfies termination, agreement, and nontriviality.*

The full proof of Theorem 5 is in Appendix C. We now present a proof overview.

Proof Overview The proof of Theorem 5 inherits heavily from the proof of Theorem 3. In fact, termination and agreement follow directly from the liveness and graph consistency of Π^G.

- **Agreement:** By Proposition 1, all honest participants output *exactly the same graph*. Therefore, to achieve one-bit agreement, the one-bit consensus output can be any fixed function of the labels that the participants output from the underlying graph protocol.
- **Termination:** By Lemma 5, honest participants' graphs grow as long as honest vertices are perpetually added. Therefore, if enough resources are allocated to honest participants, then honest participants' graphs grow to sufficient depth for them to output a bit, and Π^{bit} terminates.

To prove Theorem 5, only nontriviality remains. The intuition for the proof of nontriviality follows. We leverage the (assumed) property that honest participants have a long-term advantage in generating vertices over the corrupt participants, and run the graph consensus protocol until the graph is deep enough to guarantee that there must be substantially more honest vertices in $\mathbb{G}$ than corrupt vertices. We also use the property that each participant extracts *all* of the honest vertices in its view to guarantee that the long-term advantage in generating honest vertices translates to the fact that a majority of vertices output from each honest participant's local graph are honest. Nontriviality follows from outputting the bit that comprises the majority of one-bit labels embedded in the extracted vertices. If all honest participants have the same input b, then b is guaranteed to be the label on a majority of the extracted vertices.

The only tricky part of the proof is due to the fact that honest participants stop adding vertices below depth k^* once their local graphs become deeper than k^*, but the corrupt participants may continue to add vertices at depth k^* even after the honest participants stop adding vertices at that depth. This gives the corrupt participants extra time to add vertices with depth k^*.

We use the following technique to overcome this difficulty. Intuitively, at some time t^*, $\mathsf{D}(\mathbb{G}^{(t^*)}) - k^*$ will be so large that no vertex added at any $t > t^*$ with depth k^* will ever be extracted by any honest participant. Therefore, the extra time corrupt participants for corrupt participants to add extra vertices with depth k^* that will be output by honest participants, is limited to the range of time between t_{k^*}, defined as the moment when $\mathbb{G}$ reaches k^* depth, and t^*. Therefore, in order to ensure that the majority of vertices extracted by honest participants up to depth k^* are honest, it suffices to bound the number of corrupt vertices that can be generated in the window of time between t_{k^*} and t^*.

The proof proceeds as follows. Fist, we show that there is a distance x such that if some (corrupt) vertex v is generated at t_v and $\mathsf{D}(G_{\mathcal{H}}^{(t_v)}) - \mathsf{D}(v) > x$ then v can never be extracted from any honest participant's graph. Second, we upperbound how many corrupt vertices may be generated in any execution between the time that $\mathbb{G}$ reaches an arbitrary depth k and $G_{\mathcal{H}}$ reaches depth $k + x$, and let this number be ω. Finally, we use the honest participants' known long-term advantage to set k^* to guarantee that from the beginning of the execution until the moment when $\mathsf{D}(\mathbb{G}) = k^*$, the difference between the number of honest vertices that have been generated and the number of corrupt vertices that have been generated exceeds ω. This guarantees that when an honest participant eventually computes its output, a majority of the vertices up to depth k^* in its extracted subgraph must be honest.

References

1. Chagit Attiya, Danny Dolev, and Joseph Gil. Asynchronous byzantine consensus. In *PODC*, pages 119–133. ACM, 1984.
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. Gabriel Bracha. Asynchronous byzantine agreement protocols. *Inf. Comput.*, 75(2):130–143, 1987.
10. Gabriel Bracha and Sam Toueg. Asynchronous consensus and broadcast protocols. *Journal of the ACM (JACM)*, 32(4):824–840, 1985.
11. Ran Canetti. Universally composable security: A new paradigm for cryptographic protocols. *IACR Cryptology ePrint Archive*, 2000:67, 2000.
12. Miguel Castro, Barbara Liskov, et al. Practical byzantine fault tolerance. In *OSDI*, volume 99, pages 173–186, 1999.
13. Lin Chen, Lei Xu, Nolan Shah, Zhimin Gao, Yang Lu, and Weidong Shi. On security analysis of proof-of-elapsed-time (poet). In *International Symposium on Stabilization, Safety, and Security of Distributed Systems*, pages 282–297. Springer, 2017.
14. 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.
15. Christian Decker and Roger Wattenhofer. Information propagation in the bitcoin network. In *P2P*, pages 1–10. IEEE, 2013.
16. Danny Dolev, Cynthia Dwork, and Larry Stockmeyer. On the minimal synchronism needed for distributed consensus. *Journal of the ACM (JACM)*, 34(1):77–97, 1987.
17. Cynthia Dwork, Nancy Lynch, and Larry Stockmeyer. Consensus in the presence of partial synchrony. *Journal of the ACM (JACM)*, 35(2):288–323, 1988.
18. 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.
19. 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.

20. etherchain.org. The ethereum blockchain explorer, 2020.
21. Michael J Fischer, Nancy A Lynch, and Michael S Paterson. Impossibility of distributed consensus with one faulty process. *Journal of the ACM (JACM)*, 32(2):374–382, 1985.
22. Juan A. Garay and Aggelos Kiayias. Sok: A consensus taxonomy in the blockchain era. *IACR Cryptology ePrint Archive*, 2018:754, 2018.
23. Juan A Garay, Aggelos Kiayias, and Nikos Leonardos. The bitcoin backbone protocol: Analysis and applications. In *EUROCRYPT (2)*, pages 281–310, 2015.
24. 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.
25. 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.
26. Eleftherios Kokoris Kogias, Philipp Jovanovic, Nicolas Gailly, Ismail Khoffi, Linus Gasser, and Bryan Ford. Enhancing bitcoin security and performance with strong consistency via collective signing. In *25th USENIX Security Symposium (USENIX Security 16)*, pages 279–296, Austin, TX, August 2016. USENIX Association.
27. Leslie Lamport. Time, clocks, and the ordering of events in a distributed system. *Communications of the ACM*, 21(7):558–565, 1978.
28. Leslie Lamport et al. Paxos made simple. *ACM Sigact News*, 32(4):18–25, 2001.
29. Leslie Lamport, Robert Shostak, and Marshall Pease. The byzantine generals problem. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 4(3):382–401, 1982.
30. Silvio Micali, Salil Vadhan, and Michael Rabin. Verifiable random functions. In *Proceedings of the 40th Annual Symposium on Foundations of Computer Science*, FOCS ?99, page 120, USA, 1999. IEEE Computer Society.
31. Andrew Miller, Ari Juels, Elaine Shi, Bryan Parno, and Jonathan Katz. Permacoin: Repurposing bitcoin work for data preservation. In *Proceedings of the 2014 IEEE Symposium on Security and Privacy*, SP ?14, page 475?490, USA, 2014. IEEE Computer Society.
32. 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.
33. Andrew Miller and Joseph J LaViola Jr. Anonymous byzantine consensus from moderately-hard puzzles: A model for bitcoin. *Available on line: http://nakamotoinstitute. org/research/anonymous-byzantine-consensus*, 2014.
34. Tal Moran and Ilan Orlov. Simple proofs of space-time and rational proofs of storage. Cryptology ePrint Archive, Report 2016/035, 2016. https://eprint.iacr.org/2016/035.
35. Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system, 2008.
36. Rafael Pass, Lior Seeman, and abhi shelat. Analysis of the blockchain protocol in asynchronous networks. *IACR Cryptology ePrint Archive*, 2016:454, 2016.
37. Rafael Pass and Elaine Shi. Hybrid consensus: Efficient consensus in the permissionless model. In *LIPIcs-Leibniz International Proceedings in Informatics*, volume 91. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2017.
38. Rafael Pass and Elaine Shi. Rethinking large-scale consensus. In *Computer Security Foundations Symposium (CSF), 2017 IEEE 30th*, pages 115–129. IEEE, 2017.
39. Marshall Pease, Robert Shostak, and Leslie Lamport. Reaching agreement in the presence of faults. *Journal of the ACM (JACM)*, 27(2):228–234, 1980.
40. Team Rocket, Maofan Yin, Kevin Sekniqi, Robbert van Renesse, and Emin Gün Sirer. Scalable and probabilistic leaderless BFT consensus through metastability. *CoRR*, abs/1906.08936, 2019.
41. sawtooth.hyperledger.org. Hyperledger sawtooth poet 1.0 specification, 2020.
42. Yonatan Sompolinsky, Yoad Lewenberg, and Aviv Zohar. Spectre: A fast and scalable cryptocurrency protocol. *IACR Cryptology ePrint Archive*, 2016:1159, 2016.
43. Yonatan Sompolinsky and Aviv Zohar. PHANTOM: A scalable blockdag protocol. *IACR Cryptology ePrint Archive*, 2018:104, 2018.

A How PoX Implement Resources

In this section, we illustrate how a few of the most popular and successful forms of PoX implement resources. We stress that these illustrations are informal, and that formally showing that any scheme implements resources is a line of work that we hope we have motivated. Protocol designers should design resource schemes under assumptions that they believe to be reasonable and prove their security properties.

Common Features In every PoX mechanism, resources are hard to obtain and give special status to the messages bound to them. Protocols then base their security guarantees on the constraint that a majority of the special messages must be generated by honest participants. For each PoX that we describe, we overview how the mechanism is implemented and argue how they realize unforgeability and binding properties. We also argue that PoX implementations impose rate limits on the number of resources that are allocated over time. Note that these are sufficient properties in order to show the PoX imply consensus in our model.

A.1 Proof of Work

In a Proof of Work (PoW) scheme, processors attempt to find solutions to a hash puzzle for a cryptographic hash function H. A "solution" to a hash puzzle is a string x for which $\mathsf{H}(x) < D$, where D is a difficulty parameter. The difficulty parameter is set based on the total hash power of the network (measured as the number of hash function evaluations per second, this is an estimate of physical computing resources) in order to target a particular rate of puzzle solutions. For Bitcoin, the difficulty parameter is set such that a puzzle solution is found about every 10 minutes [35]; in Ethereum the difficulty parameter is set such that a puzzle solution is found about every 13 seconds [20].

In most Proof of Work schemes, the input x is composed of a nonce, a payload, and a pointer to a previous puzzle solution. When a puzzle solution is found, we consider the input x to be the string that is bound to the resource. Note that to strictly implement resources, it is not necessary that an input to a hash puzzle include a pointer to a previous hash puzzle; however, this property is used by many Proof-of-Work protocols to enforce a graph structure.

Unforgeability of a resource in a proof of work scheme follows from the hardness and verifiability of the hash function. Honest parties can easily verify that a string x is a valid solution to the hash puzzle $\mathsf{H}(\cdot) < D$, and it should be hard to find a puzzle solution without evaluating the hash puzzle (see [32] for a discussion).

Binding of a resource follows from the collision resistance of the hash function. Given a resource bound to string x, in order to claim the resource has been bound to another string, a corrupt party must find an x' such that $\mathsf{H}(x') = \mathsf{H}(x)$.

Note that proof-of-work schemes change their difficulty parameter on a regular basis based on the hash rate of the network in order to maintain a particular rate at which solutions are found. In order to show that a proof-of-work scheme implements resources, one would have to identify realistic assumptions from which to show that difficulty calibration of the hash puzzle effectively upperbounds the rate at which hash puzzles are found.

A.2 Proof of Stake

Proof of Stake (PoS) schemes implement resources as binding lottery tickets. At each time step, each participant evaluates some number of virtual lottery tickets to determine if it is the leader in the blockchain protocol at that time step. The number of lottery tickets each processor can evaluate at a time step is proportional to its stake in the system at that time. Each Proof of Stake protocol is parameterized in order to upperbound the number of winning lottery tickets that are evaluated per time step; this mechanism implements rate-limiting. (This is analogous to parameterizing the number of proof of work solutions per network delay, or upperbounding the number of resources that are allocated per span of time.) We remark that in each of the schemes we overview, the schemes rely on either knowing the maximum communication delay of the network or on loosely synchronized clocks in order to synchronize the rounds of the lottery.

Lottery by VRF In the PoS schemes of both Ouroboros [14] and Algorand [24], lottery tickets are implemented using a verifiable random function (VRF) [30, 14]. A participant evaluates a lottery ticket by computing $\mathsf{vrf.prove}_{\mathsf{sk}}(x) \to \pi$, where sk is the secret key associated with a stake in the system that the participant owns a particular time, and π is the output of the vrf. The VRF input x encodes a time slot and a nonce (where the nonce encodes some global random state). A lottery ticket is considered to be a "winner" if $\pi < D$, for some tunable parameter D. The other participants verify the role of a claimed leader using the algorithm $\mathsf{vrf.vfy}_{\mathsf{pk}}(x, \pi)$, where pk is the public key associated with sk. While a participant is a leader, it signs its messages using sk to refer to its winning lottery ticket for authority, and participants use pk to verify the signatures. (We are eliding details about the cryptographic sortition implemented by Algorand.)

We consider the message bound to the resource implemented by PoS lottery to be the pair (x, pk), where x is the input to the VRF and pk is the public key that verifies the VRF proof. Unforgeability follows from the unpredictability and verifiability of the VRF. Specifically, no adversary should not be able to efficiently "predict" keys which will be leaders in any particular time slot, and any claimed winner is verifiable by the honest participants. Binding follows from the unpredictability of the VRF. Given one solution, it should be hard to find another input to the VRF that evaluates to the same proof.

In order to show that VRF-based PoS implements resources, one must argue that no corrupt participant can increase its share of VRF puzzle solutions to be more than roughly its proportion of stake in the system. We refer to Ouroboros [14] and Algorand [24] for in-depth discussions. In particular, refer to the discussion in Ouroboros on VRF Unpredictability under Malicious Key Generation ([14] Section 3.2), and in Algorand refer to the discussions about choosing the VRF seed in each round and setting secret keys well before each round ([24] Sections 5.1, 5.2).

Lottery by Hash Function The Proof of Stake mechanism by Snow White [8] differs from Ouroboros and Algorand in that it does not use a VRF. Instead, it uses a cryptographic hash function that is seeded by a stateful nonce that depends on the previous hash puzzle solutions. Specifically, the a proof of stake is evaluated as $\mathsf{H}(r, \mathsf{pk}, t) < D$, where r is a stateful nonce, pk is a public key for a digital signature scheme, and t is a timestamp. If the output of the hash function is less than the difficulty parameter D, then the participant with public key pk becomes a leader, and signs its messages with the secret key corresponding to pk while it is a leader. We therefore consider pk to be the string bound to the resource implemented by Snow White's proof of stake.

We refer to the discussion in Snow White for a detailed treatment about how the protocol constrains the adversary's proportion of Proof of Stake puzzles proportional to not much more

than the proportion of its stake in the system. They reason about security of their proof of stake mechanism under adversarially based hashes in [8] Sections 2 and G.

A.3 Non-Cryptographic PoX

There have been many additional cryptographic PoX variants proposed, for example Proof of Spacetime [34] and Proof of Retrievability [31]. We do not analyze them all here. However, we do remark that PoX need not necessarily be implemented using cryptography. For example, Proof of Elapsed Time [41, 13] elects leaders in a consensus protocol via verifiable timer. Additionally, we believe that resources could be implemented in low-power environments in which participants seldom have enough energy to send a message. In this case, every message would be associated with a resource, as the resource represents physical energy.

We believe that future research could study ways to move resource allocation to the environment (for example, by random lottery based on external factors), rather than by solving hash puzzles.

B Proof of Graph Consensus Protocol Π^G

We present the proof of Theorem 3, which proves graph consistency and liveness for our graph consensus protocol Π^G.

Theorem 3. *For 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 $f(N,\alpha,\varepsilon,\rho) = h(N,\alpha,\varepsilon,\rho) = \alpha N - \varepsilon - \rho(\ell^*+1)$.*

Most of our effort towards proving Theorem 3 is focused on the proof of Proposition 1, which we restate here.

Proposition 1. *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 $\mathsf{extract}(G_p^{(t)})\lfloor_k = \mathsf{extract}(G_q^{(t')})\lfloor_k$.*

Consistency will follow directly, and liveness will follow easily from the techniques we use for Proposition 1. We begin to prove Proposition 1 by showing consistency of the honest vertices that honest participants output. Recall for the duration of the proof that we require $\alpha > \rho\beta c$, where β and c are defined as in the protocol specification; particularly, $\beta = 1-\alpha$ is the long-term proportion of corrupt resources and c is a derived constant.

Recall that in Section 5.3, we presented an overview of the proof of Proposition 1. We reproduce these lemmas here and then present their proofs.

Lemma 1 proves that the honest vertices in honest participants' extracted graphs are consistent.

Lemma 1 (Honest Vertex Extraction). *For every time t, honest participant p active at t, and honest vertex $v \in \mathbb{G}^{(t)}$:*

$$\mathsf{D}(G_p^{(t)}) - \mathsf{D}(v) > \ell_1 \implies v \in \mathsf{extract}(G_p^{(t)})$$

We prove it by decomposition into Lemmas 2 and 3

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

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

We then prove consistency of corrupt vertices by proving Lemma 4.

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 $\mathsf{D}(u) - \mathsf{D}(v) \leq \ell_2$.*

In section B.1 we prove Lemma 2. In section B.2 we prove Lemma 3 and complete the proof of Lemma 1. In Section B.3 we prove Lemma 4. Finally, in Section B.4 we conclude the proofs of Proposition 1 and Theorem 3.

B.1 Consistency of Views for Honest Participants

Towards proving Lemma 2, we begin our technical lemmas with a foundational statement that lowerbounds the growth rate of the depth of $\mathbb{G}$ in an execution as a function of the number of vertices that are generated.

Intuitively, the depth of $\mathbb{G}$ is driven up by honest participants which add vertices that increase the depths of their local graphs. As a tool to understand what vertices *must* be in a participant's local graph at any point in time, we define a virtual graph $G_{\mathcal{H}}^{(t)}$, which for time t answers "what is the *smallest* graph of an honest participant at time t?" One may consider that $G_{\mathcal{H}}^{(t)}$ is guaranteed to contain *at least* all of the honest vertices that are generated before $t - \Delta$, since each honest vertex is immediately multicast when it is generated, and at most Δ time may elapse before the multicast message is guaranteed to be delivered.

The following lemma lowerbounds the growth of $G_{\mathcal{H}}$ between any t and $t' > t$ as a function of the number of resources allocated between t and t'. The growth of $G_{\mathcal{H}}$ is lowerbounded by the number of resources that are allocated to honest participants and by how many honest resources can be allocated concurrently. $G_{\mathcal{H}}$ must grow by at least 1 depth for every ρ honest vertices that are generated. This is because at most ρ honest participants can concurrently generate vertices with the same depth before one of their vertices is guaranteed to be delivered, and increases the depth of all honest graphs that have not yet reached that depth.

Lemma 5 (Lowerbound Honest Growth). *Define $G_{\mathcal{H}}^{(t)}$ as:*

$$G_{\mathcal{H}}^{(t)} = \bigcap_{t' \geq t, p \text{ active at } t'} G_p^{(t')}$$

For all t and $t' > t$: $\mathsf{D}(G_{\mathcal{H}}^{(t')}) \geq \mathsf{D}(G_{\mathcal{H}}^{(t)}) + \frac{\alpha|\Psi^{(t,t')}| - \varepsilon - \rho}{\rho}$.

Proof. Assume towards contradiction that for some t and t' in an execution, $\mathsf{D}(G_{\mathcal{H}}^{(t')}) - \mathsf{D}(G_{\mathcal{H}}^{(t)}) < \frac{\alpha|\Psi^{(t,t')}| - \varepsilon - \rho}{\rho}$. The lemma follows from the following three claims.

Claim 1 *Between times t and t' in any execution, at least $\alpha|\Psi^{(t,t')}| - \varepsilon - \rho$ honest vertices generated between t and t' are in $G_{\mathcal{H}}^{(t')}$.*

Proof. Consider an execution between times t and t'. By α, ε-honest execution (Definition 9), at least $\alpha|\Psi^{(t,t')}| - \varepsilon$ resources are allocated to honest participants between t and t'. Recall that by the protocol specification, whenever an honest participant receives a resource, it immediately generates

and multicasts a new vertex. The only reason why an honest vertex may not be in $G_p^{(t')}$ for any participant p active at t' is if the vertex is delayed over the network; therefore, only vertices generated after $t' - \Delta$ may not be in $G_{\mathcal{H}}^{(t')}$. By ρ-rate-limiting (Definition 7), at most ρ resources may be allocated between $t' - \Delta$ and t'. Therefore, at least $\alpha|\Psi^{(t,t')}| - \varepsilon - \rho$ honest vertices generated between t and t' are in $G_{\mathcal{H}}^{(t')}$. □

Claim 2 *For any time t in an execution, every honest vertex generated after t has depth greater than $\mathsf{D}(G_{\mathcal{H}}^{(t)})$.*

Proof. Recall by the protocol specification, whenever an honest participant p generates a vertex v at time $s > t$, v is the unique deepest vertex in $G_p^{(s)}$. Thus $\mathsf{D}(v) = \mathsf{D}(G_p^{(s)})$. Additionally, observe that v cannot be in $G_{\mathcal{H}}^{(t)}$ since it cannot be delivered to all honest participants by t if it is generated at $s > t$. Because $G_{\mathcal{H}}^{(t)} \subseteq G_p^{(s)}$ by definition, v is the unique deepest vertex in $G_p^{(s)}$, and $v \notin G_{\mathcal{H}}^{(t)}$, it follows immediately that $\mathsf{D}(G_p^{(s)}) \geq \mathsf{D}(G_{\mathcal{H}}^{(t)}) + 1$, and therefore $\mathsf{D}(v) > \mathsf{D}(G_{\mathcal{H}}^{(t)})$. □

Claim 3 *For any time t in an execution, there may be at most ρ honest vertices in $\mathbb{G}^{(t)}$ with the same depth.*

Proof. Recall by the protocol specification, whenever an honest participant p generates a vertex, the generated vertex is the unique deepest vertex in p's graph. Therefore, if an honest participant generates a vertex of depth d, then before it generated the vertex, its graph had depth $d - 1$. It follows that if more than ρ honest vertices with depth d are generated, then there must be more than ρ honest participants which, when allocated a resource, have graphs of depth $d - 1$. Let p_1, $\ldots$, $p_{\rho+1}$, be the first participants, in order, which generate vertices when their local graphs have depth $d - 1$. Let v_1, $\ldots$, $v_{\rho+1}$ be the vertices that they generate, and let the vertices be generated at t_{v_1}, $\ldots$, $t_{v_{\rho+1}}$, respectively.

It must be that $t_{v_{\rho+1}} > t_{v_1} + \Delta$ because of ρ-rate limiting (Definition 7). But this implies that v_1 must be in $G_{p_{\rho+1}}^{(t_{v_{\rho+1}})}$, and therefore $\mathsf{D}(G_{p_{\rho+1}}^{(t_{v_{\rho+1}})}) \geq d$. This is a contradiction. □

We now conclude the proof of the lemma. By Claim 1, at least $\alpha|\Psi^{(t,t')}| - \varepsilon - \rho$ of the vertices which are allocated between t and t' are in $G_{\mathcal{H}}^{(t')}$. By Claim 2, all such vertices have depth greater than $\mathsf{D}(G_{\mathcal{H}}^{(t)})$. By the contradiction hypothesis, $\mathsf{D}(G_{\mathcal{H}}^{(t')}) - \mathsf{D}(G_{\mathcal{H}}^{(t)}) < \frac{\alpha|\Psi^{(t,t')}| - \varepsilon - \rho}{\rho}$. Therefore, there must be some depth $d > \mathsf{D}(G_{\mathcal{H}}^{(t)})$ such that more than ρ honest vertices in $\mathbb{G}^{(t')}$ have depth d. This is a contradiction with Claim 3. □

Next, we present a lemma that bounds the difference between the depth of $\mathbb{G}^{(t)}$ and the depth of $G_p^{(t)}$ for any honest participant p active at any time t. Intuitively, this bounds how far behind $\mathbb{G}$ that an honest participant's view can lag at any point in time.

Lemma 6 (Bounding $\mathsf{D}(G_p^{(t)})$ relative to $\mathsf{D}(\mathbb{G}^{(t)})$). *Let $\gamma = (1+\beta)\rho + \varepsilon + \frac{\varepsilon}{\rho} + 1$. If $\frac{\alpha}{\rho} > \beta$, then for all t and honest participant p active at t, $\mathsf{D}(\mathbb{G}^{(t)}) - \mathsf{D}(G_p^{(t)}) \leq \gamma$.*

Sketch. The proof technique is to select an honest vertex v_c in reference to which the growth of both $\mathbb{G}^{(t)}$ and $G_p^{(t)}$ can be measured. We then upperbound the difference $\mathsf{D}(\mathbb{G}^{(t)}) - \mathsf{D}(G_p^{(t)})$ by upperbounding $\mathsf{D}(\mathbb{G}^{(t)}) - \mathsf{D}(v_c)$ and lowerbounding $\mathsf{D}(G_p^{(t)}) - \mathsf{D}(v_c)$. The crux of the proof is to show

that there must exist a vertex v_c with respect to which the growth of each graph can be measured. Given the existence of v_c, we can bound the differences $\mathsf{D}(\mathbb{G}^{(t)}) - \mathsf{D}(v_c)$ and $\mathsf{D}(G_p^{(t)}) - \mathsf{D}(v_c)$ in terms of the number of vertices that have been generated between the time when v_c is generated and t. We then use these bounds to show the desired statement.

Proof. Assume for the sake of reaching a contradiction that in some execution at time t and for some participant p active at t

$$\mathsf{D}(\mathbb{G}^{(t)}) - \mathsf{D}(G_p^{(t)}) > \gamma \tag{1}$$

Let v_d be the vertex in $\mathsf{D}(\mathbb{G}^{(t)})$ with the greatest depth. (If there are multiple such vertices, choose any one as v_d.) Choose any *longest path* in $\mathbb{G}^{(t)}$ from root to v_d, which is defined to be a path $\mathsf{root} \to v_d$ such that the depth spanned by every edge is 1. Note that such a path *must* exist, since the depth of each vertex is defined to be one more than its deepest predecessor, and it is therefore always possible to walk backwards from v_d to root via a path in which each edge spans depth 1. Let v_c be the honest vertex with the maximum depth on this path subject to the constraint that v_c is in $G_{\mathcal{H}}^{(t)}$; in the worst case (if no honest vertices on the path are in $G_{\mathcal{H}}^{(t)}$), $v_c = \mathsf{root}$. We let t_{v_c} denote the time at which v_c is generated. (If $v_c = \mathsf{root}$, then let $t_{v_c} = 0$.)

We upperbound $\mathsf{D}(\mathbb{G}^{(t)}) - \mathsf{D}(G_p^{(t)})$, using v_c as a reference point, by first decomposing it into parts

$$\mathsf{D}(\mathbb{G}^{(t)}) - \mathsf{D}(G_p^{(t)}) = [\mathsf{D}(\mathbb{G}^{(t)}) - \mathsf{D}(v_c)] - [\mathsf{D}(G_p^{(t)}) - \mathsf{D}(v_c)] \tag{2}$$

It will suffice to upperbound $\mathsf{D}(\mathbb{G}^{(t)}) - \mathsf{D}(v_c)$ and to lowerbound $\mathsf{D}(G_p^{(t)}) - \mathsf{D}(v_c)$. We begin with an upperbound for $\mathsf{D}(\mathbb{G}^{(t)}) - \mathsf{D}(v_c)$:

Claim 4

$$\mathsf{D}(\mathbb{G}^{(t)}) - \mathsf{D}(v_c) \leq \beta|\Psi^{(t_{v_c},t)}|+\varepsilon + \rho \tag{3}$$

Proof. Recall that because v_d is the deepest vertex in $\mathbb{G}^{(t)}$, $\mathsf{D}(v_d) = \mathsf{D}(\mathbb{G}^{(t)})$ by definition. We upperbound $\mathsf{D}(\mathbb{G}^{(t)}) - \mathsf{D}(v_c)$ by upperbounding $\mathsf{D}(v_d) - \mathsf{D}(v_c)$. Recall also that there must be a path $v_c \to v_d$ on which each edge of the path spans 1 depth. To upperbound $\mathsf{D}(v_d) - \mathsf{D}(v_c)$, it therefore suffices to upperbound the number of vertices on the path $v_c \to v_d$. We divide the analysis into two parts: first we upperbound the number of honest vertices on the path, and then we upperbound the number of corrupt vertices on the path.

We claim that there may be at most ρ honest vertices on the path $v_c \to v_d$. Recall that v_c is defined to be the deepest vertex on a longest path from root to v_d which is also in $G_{\mathcal{H}}^{(t)}$. All of the honest vertices on the path (which are successors of v_c) must not be in $G_{\mathcal{H}}^{(t)}$. In order for an honest vertex v to not be in $G_{\mathcal{H}}^{(t)}$, there must be some honest participant q activated at some $t' \geq t$ for which v is not in $G_q^{(t')}$. This is only possible if v is delayed over the network to q at t'; therefore, any honest vertex which is on the path $v_c \to v_d$ but is not in $G_{\mathcal{H}}^{(t)}$ must have been generated after $t - \Delta$ (by Definition 3). By the limit on the rate of resource allocations per Δ time (Definition 7), there may be at most ρ such honest vertices. Therefore, there may be at most ρ honest vertices on the path $v_c \to v_d$.

We now upperbound the number of corrupt vertices on the path $v_c \to v_d$. Consider that all corrupt vertices on the path $v_c \to v_d$ must be generated after t_{v_c} because each is a successor of v_c. By the definition of a β, ε-corrupt execution (Definition 9), at most $\beta|\Psi^{(t_{v_c},t)}|+\varepsilon$ corrupt resources

may be generated between t_{v_c} and t. It follows that there are at most $\beta|\Psi^{(t_{v_c},t)}|+\varepsilon$ corrupt vertices on the path $v_c \to v_d$.

Summing the upperbounds for honest and corrupt vertices on the path $v_c \to v_d$, it follows that

$$\mathsf{D}(\mathbb{G}^{(t)}) - \mathsf{D}(v_c) \leq \beta|\Psi^{(t_{v_c},t)}|+\varepsilon+\rho$$

as claimed. □

We next lowerbound $\mathsf{D}(G_p^{(t)}) - \mathsf{D}(v_c)$:

Claim 5

$$\mathsf{D}(G_p^{(t)}) - \mathsf{D}(v_c) \geq \frac{\alpha|\Psi^{(t_{v_c}+\Delta,t)}|-\varepsilon-\rho}{\rho} \tag{4}$$

Proof. Lowerbounding the difference $\mathsf{D}(G_p^{(t)}) - \mathsf{D}(v_c)$ is challenging because we do not have enough information about v_c to directly upperbound its depth. However, we do know that $\mathsf{D}(G_{\mathcal{H}}^{(t_{v_c}+\Delta)}) \geq \mathsf{D}(v)$, since v must be in the view of every honest participant activated at or after $t_{v_c}+\Delta$, and by definition it must therefore be in $G_{\mathcal{H}}^{(t_{v_c}+\Delta)}$.

Given the upperbound of $\mathsf{D}(v_c)$ in terms of $G_{\mathcal{H}}^{(t_{v_c}+\Delta)}$, we complete the desired lowerbound of $\mathsf{D}(G_p^{(t)}) - \mathsf{D}(v_c)$ by lowerbounding $G_p^{(t)}$ in terms of $G_{\mathcal{H}}^{(t)}$ and directly invoking Lemma 5. This is trivial, since we know $G_{\mathcal{H}}^{(t)} \subseteq G_p^{(t)}$ by definition, and therefore $\mathsf{D}(G_p^{(t)}) \geq \mathsf{D}(G_{\mathcal{H}}^{(t)})$.

We conclude:

$$\begin{aligned}
\mathsf{D}(G_p^{(t)}) - \mathsf{D}(v_c) &\geq \mathsf{D}(G_p^{(t)}) - \mathsf{D}(G_{\mathcal{H}}^{(t_{v_c}+\Delta)}) \\
&\geq \mathsf{D}(G_{\mathcal{H}}^{(t)}) - \mathsf{D}(G_{\mathcal{H}}^{(t_{v_c}+\Delta)}) \\
&\geq \frac{\alpha|\Psi^{(t_{v_c}+\Delta,t)}|-\varepsilon-\rho}{\rho}
\end{aligned}$$

□

We now use the upperbound on $\mathsf{D}(\mathbb{G}^{(t)}) - \mathsf{D}(v_c)$ and the lowerbound on $\mathsf{D}(G_p^{(t)}) - \mathsf{D}(v_c)$ to conclude the lemma. Recalling (in order) Inequality 1, Equation 2, Inequality 3 and Inequality 4, we conclude:

$$\begin{aligned}
\gamma &< \mathsf{D}(\mathbb{G}^{(t)}) - \mathsf{D}(G_p^{(t)}) \\
&= [\mathsf{D}(\mathbb{G}^{(t)}) - \mathsf{D}(v_c)] - [\mathsf{D}(G_p^{(t)}) - \mathsf{D}(v_c)] \\
&\leq [\beta|\Psi^{(t_{v_c},t)}|+\varepsilon+\rho] - [\frac{\alpha|\Psi^{(t_{v_c}+\Delta,t)}|-\varepsilon-\rho}{\rho}] \\
&= \beta|\Psi^{(t_{v_c},t_{v_c}+\Delta)}|+\beta|\Psi^{(t_{v_c}+\Delta,t)}|+\varepsilon+\rho+\frac{\varepsilon}{\rho}+1-\frac{\alpha|\Psi^{(t_{v_c}+\Delta,t)}|}{\rho} \\
&\leq (\beta-\frac{\alpha}{\rho})|\Psi^{(t_{v_c}+\Delta,t)}|+(1+\beta)\rho+\varepsilon+\frac{\varepsilon}{\rho}+1
\end{aligned}$$

where the last inequality follows because $\beta|\Psi^{(t_{v_c},t_{v_c}+\Delta)}| \leq \beta\rho$, since at most ρ resources may be allocated between t_{v_c} and $t_{v_c}+\Delta$ by the rate limit on resource allocations (Definition 7).

Therefore, it must be the case that

$$(\beta - \frac{\alpha}{\rho})|\Psi^{(t_{v_c}+\Delta,t)}|+(1+\beta)\rho + \varepsilon + \frac{\varepsilon}{\rho} + 1 > \gamma \tag{5}$$

but when $\frac{\alpha}{\rho} > \beta$, this is true only when $|\Psi^{(t_{v_c}+\Delta,t)}|$ is negative. This is a contradiction because there cannot be negative resource allocations. □

We now complete the proof of Lemma 2, which we restate here. Intuitively, the lemma shows that if for some honest participant p active at time t, and some honest vertex v, $\mathsf{D}(G_p^{(t)}) - \mathsf{D}(v) > \ell_1 = \rho + \gamma$, then more than Δ time has elapsed since v was generated and multicast. It will follow that $v \in G_p^{(t)}$.

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

Proof. Assume that there is a vertex v generated by an honest participant q at time t_v, and there is another honest participant p active at time $t > t_v$ such that $v \notin G_p^{(t)}$. We will show that it must be the case that $\mathsf{D}(G_p^{(t)}) - \mathsf{D}(v) \leq \ell_1$.

Consider that because $G_p^{(t)} \subseteq \mathbb{G}^{(t)}$, the difference $\mathsf{D}(G_p^{(t)}) - \mathsf{D}(v)$ is trivially upperbounded by $\mathsf{D}(\mathbb{G}^{(t)}) - \mathsf{D}(v)$. The difference $\mathsf{D}(\mathbb{G}^{(t)}) - \mathsf{D}(v)$ can be decomposed into the sum of two parts: $\mathsf{D}(\mathbb{G}^{(t_v)}) - \mathsf{D}(v)$, the difference in depth between v and $\mathbb{G}$ at the moment when v is generated, and $\mathsf{D}(\mathbb{G}^{(t)}) - \mathsf{D}(\mathbb{G}^{(t_v)})$, or the amount that $\mathbb{G}$ has grown since v was generated.

First, we observe that $\mathsf{D}(\mathbb{G}^{(t_v)}) - \mathsf{D}(v) = \mathsf{D}(\mathbb{G}^{(t_v)}) - \mathsf{D}(G_q^{(t_v)})$, since v is the deepest vertex in $G_q^{(t_v)}$. We directly apply Lemma 6 to bound $\mathsf{D}(\mathbb{G}^{(t_v)}) - \mathsf{D}(G_q^{(t_v)}) \leq \gamma$.

Second, we upperbound $\mathsf{D}(\mathbb{G}^{(t)}) - \mathsf{D}(\mathbb{G}^{(t_v)})$ as follows. Recall that when an honest participant generates a vertex, it immediately multicasts the vertex. If v is not in $G_p^{(t)}$, then it must be delayed over the network; therefore, it must be that $t < t_v + \Delta$. We use the rate limit on resource allocations (Definition 7) to conclude that $|\Psi^{(t_v,t)}| \leq \rho$. Because $\mathbb{G}$ can increase in depth between t_v and t by at most the number of vertices which are generated between t_v and t, it follows that $\mathsf{D}(\mathbb{G}^{(t)}) - \mathsf{D}(\mathbb{G}^{(t_v)}) \leq |\Psi^{(t_v,t)}| \leq \rho$.

Therefore we conclude,

$$\begin{aligned}
\mathsf{D}(G_p^{(t)}) - \mathsf{D}(v) &\leq \mathsf{D}(\mathbb{G}^{(t)}) - \mathsf{D}(v) \\
&= \mathsf{D}(\mathbb{G}^{(t)}) - \mathsf{D}(\mathbb{G}^{(t_v)}) + \mathsf{D}(\mathbb{G}^{(t_v)}) - \mathsf{D}(G_q^{(t_v)}) \\
&\leq |\Psi^{(t_v,t)}| + \gamma \\
&\leq \rho + \gamma \\
&= \ell_1
\end{aligned}$$

□

B.2 Outputting Consistent Honest Vertices

We now re-state and prove Lemma 3, which states that an honest participant always extracts every honest vertex in its local graph.

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

We show that every vertex in an honest participant's local graph must be either a starting vertex in the graph or a predecessor of a starting vertex in the graph. We prove this in two steps. First we show that every vertex v that is generated by an honest participant is guaranteed to gain an honest successor in $\mathbb{G}$ which is at most c deeper than v. Second, we show that if an honest vertex v is more than $c+\rho$ depth from the end of an honest participant's graph, then its guaranteed honest successor must also be in the graph. Recall that the starting vertices in a participant's graph are defined to be those with depth within $c + \rho$ of the graph itself. It follows that from every honest vertex which is not a starting vertex in a participant's graph, there must be a path to an honest starting vertex in the graph.

Before we proceed, we first introduce a useful property of an execution that bounds how many consecutive corrupt vertices may be generated in a span of time in which no honest vertices are generated.

Fact 1. *For all t, t' in an α, ε-honest execution: if $|\Psi_{\mathsf{hon}}^{(t,t')}| = 0$ then $|\Psi^{(t,t')}| \leq \frac{\varepsilon}{\alpha}$.*

Proof. Direct from Definition 9. $|\Psi_{\mathsf{hon}}^{(t,t')}|$ is lower bounded by $\alpha|\Psi^{(t,t')}| - \varepsilon$, which is greater than 0 for all t, t' for which $|\Psi^{(t,t')}| > \frac{\varepsilon}{\alpha}$. □

Next we show that each honest vertex v is guaranteed to gain at least one honest vertex as a successor in $\mathbb{G}$ before $\mathbb{G}$ grows too far away from v. Specifically, we show that maximum the difference in depth between v and its guaranteed honest successor is c, and that at any time t after v is generated, if $\mathsf{D}(\mathbb{G}^{(t)}) - \mathsf{D}(v) > c$ then v's honest successor is guaranteed to already exist in $\mathbb{G}$.

Lemma 7. *Let $c = \gamma + \rho + \frac{\varepsilon}{\alpha}$. For every time t and honest vertex $v \in \mathbb{G}^{(t)}$, $\mathsf{D}(\mathbb{G}^{(t)}) - \mathsf{D}(v) > c$ implies there exists an honest vertex u in $\mathbb{G}^{(t)}$ such that $\mathsf{D}(u) - \mathsf{D}(v) \leq c$ and u is reachable from v.*

Sketch. Let t_v be the time when v is generated. First, we show that if $\mathsf{D}(\mathbb{G}^{(t)}) - \mathsf{D}(v) > c$, then there must be some honest vertex generated after $t_v + \Delta$. Let v_1 be the first honest vertex generated after $t_v + \Delta$. Second, we show that $\mathsf{D}(v_1) - \mathsf{D}(v) \leq c$, and that v_1 is reachable from v.

Proof. For a vertex u, use the notation that t_u is the time at which u is generated. The proof follows from the following two claims.

Claim 6 *If $\mathsf{D}(\mathbb{G}^{(t)}) - \mathsf{D}(v) > c$, then there must be an honest vertex generated after $t_v + \Delta$.*

Proof. Assume that $\mathsf{D}(\mathbb{G}^{(t)}) - \mathsf{D}(v) > c$ but there is no honest vertex generated after $t_v + \Delta$.

Consider that

$$\mathsf{D}(\mathbb{G}^{(t)}) - \mathsf{D}(v) = \mathsf{D}(\mathbb{G}^{(t)}) - \mathsf{D}(\mathbb{G}^{(t_v)}) + \mathsf{D}(\mathbb{G}^{(t_v)}) - \mathsf{D}(v)$$

Lemma 6 immediately bounds $\mathsf{D}(\mathbb{G}^{(t_v)}) - \mathsf{D}(v) \leq \gamma$. If $\mathsf{D}(\mathbb{G}^{(t)}) - \mathsf{D}(v) > c$ and $\mathsf{D}(\mathbb{G}^{(t_v)}) - \mathsf{D}(v) \leq \gamma$, then it must be the case that $\mathsf{D}(\mathbb{G}^{(t)}) - \mathsf{D}(\mathbb{G}^{(t_v)}) > \rho + \frac{\varepsilon}{\alpha}$. This immediately implies that $|\Psi^{(t_v,t)}| > \rho + \frac{\varepsilon}{\alpha}$, because $\mathbb{G}$ cannot grow in depth between t_v and t more than the number of vertices which are generated in that time.

Let $v_1, \ldots, v_{\rho+\frac{\varepsilon}{\alpha}+1}$, be in chronological order the first $\rho + \frac{\varepsilon}{\alpha} + 1$ vertices generated between t_v and t. Consider that between t_v and $t_v + \Delta$, as most ρ vertices may have been generated because

of the rate limit on vertex generation (Definition 7). It follows that $v_{\rho+1}, \ldots, v_{\rho+\frac{\varepsilon}{\alpha}+1}$ must all be generated after $t_v + \Delta$. Moreover, by the contradiction hypothesis, they are all corrupt. But this means that more than $\frac{\varepsilon}{\alpha}$ consecutive corrupt vertices are generated, which is a contradiction to Fact 1. □

Next, using many of the same techniques, we bound the difference in depth between v and this honest vertex generated after $t_v + \Delta$, and show that it is reachable from v.

Claim 7 *In any execution, consider any honest vertex v for which some honest vertex is generated after $t_v + \Delta$, and let v_1 be the first vertex generated after $t_v + \Delta$. Then $\mathsf{D}(v_1) - \mathsf{D}(v) \leq c$ and v_1 is reachable from v.*

Proof. First we show that $\mathsf{D}(v_1) - \mathsf{D}(v) \leq c$. Assume that it $\mathsf{D}(v_1) - \mathsf{D}(v) > c$.

As in the previous claim, we observe that $\mathsf{D}(v_1) - \mathsf{D}(v)$ is upperbounded by the difference in depth between v and $\mathbb{G}$ at the moment that v is generated, plus the amount that $\mathbb{G}$ grows between t_v and t_{v_1}. Specifically,

$$\begin{aligned}\mathsf{D}(v_1) - \mathsf{D}(v) &= \mathsf{D}(v_1) - \mathsf{D}(\mathbb{G}^{(t_v)}) + \mathsf{D}(\mathbb{G}^{(t_v)}) - \mathsf{D}(v) \\ &\leq \mathsf{D}(\mathbb{G}^{(t_{v_1})}) - \mathsf{D}(\mathbb{G}^{(t_v)}) + \mathsf{D}(\mathbb{G}^{(t_v)}) - \mathsf{D}(v)\end{aligned}$$

First, by an immediate application of Lemma 6, $\mathsf{D}(\mathbb{G}^{(t_v)}) - \mathsf{D}(v) \leq \gamma$. Second we bound how much $\mathbb{G}$ can grow between t_v and t_{v_1}. Clearly, $\mathsf{D}(\mathbb{G}^{(t_{v_1})}) - \mathsf{D}(\mathbb{G}^{(t)}) \leq |\Psi^{(t,t_{v_1})}|$, since $\mathbb{G}$ cannot grow by more vertices than the number of resources allocated in this span of time.

As in the previous claim, it must be the case that $|\Psi^{(t,t_{v_1})}| > \rho + \frac{\varepsilon}{\alpha}$. By an analogous argument to the previous claim, since only ρ vertices may be generated between t and $t+\Delta$, this implies that $|\Psi^{(t_v+\Delta,t_{v_1})}| > \frac{\varepsilon}{\alpha}$. But because v_1 is the *first* honest vertex generated after $t_v + \Delta$, this leads to the conclusion that more than $\frac{\varepsilon}{\alpha}$ consecutive corrupt vertices are generated between $t_v + \Delta$ and t_{v_1}, which is a contradiction with Fact 1. We therefore conclude that $\mathsf{D}(v_1) - \mathsf{D}(v) \leq c$.

Next we show that v_1 is reachable from v. Let r be the participant that generates v_1. We claim that v must be in $G_r^{(t_{v_1})}$. Recall that v is generated by an honest participant and immediately multicast at t_v. Therefore, v must be in the local graph of every honest participant activated after $t_v + \Delta$. Because $t_{v_1} > t_v + \Delta$, it is immediate that v is in $G_r^{(t_{v_1})}$.

Consider that if v has outdegree 0 in r's local graph before adding t_{v_1}, then because $\mathsf{D}(v_1) - \mathsf{D}(v) \leq c$, the protocol specification requires that r add an edge from v to v_1. If $\mathsf{outdegree}(v) > 0$ in r's local graph before adding t_{v_1}, then there must be some vertex u in r's local graph with an edge from v. If $\mathsf{outdegree}(u) > 0$ in r's local graph, then recursively follow u's successors until reaching a vertex w that has outdegree 0 in r's view when r generates v_1. Notice that because w is a successor of v, $\mathsf{D}(w) > \mathsf{D}(v)$, and it follows that $\mathsf{D}(w) - \mathsf{D}(v_1) < c$. Therefore, by the protocol specification, r must add an edge from w to v_1, and there is a path from v to v_1. □

The lemma follows immediately by composing the two claims. By Claim 6 $\mathsf{D}(\mathbb{G}^{(t)}) - \mathsf{D}(v) > c$ implies that there is an honest vertex generated after $t_v + \Delta$. By Claim 7, the first honest vertex v_1 generated after $t_v + \Delta$ must be reachable from v and $\mathsf{D}(v_1) - \mathsf{D}(v) \leq c$. □

The previous lemma showed that each honest vertex v is guaranteed to gain an honest successor in $\mathbb{G}$ before $\mathbb{G}$ grows to be much deeper than v. However, although v's honest successor is guaranteed to exist in $\mathbb{G}^{(t)}$ if $\mathsf{D}(\mathbb{G}^{(t)}) - \mathsf{D}(v) > c$, it is not necessarily true that the honest successor is in

$G_p^{(t)}$ if $\mathsf{D}(G_p^{(t)}) - \mathsf{D}(v) > c$. In the following lemma we show that instead, we can guarantee that if $\mathsf{D}(G_p^{(t)}) - \mathsf{D}(v) > c + \rho$, then v is guaranteed to have at least one honest successor in $G_p^{(t)}$. Intuitively, the extra ρ required to show the statement for honest participants' local graphs allows enough time for v's honest successor v_1 to be delivered over the network to every honest participant.

Lemma 8. *For every time t, honest participant p active at t, and honest vertex $v \in G_p^{(t)}$: $\mathsf{D}(G_p^{(t)}) - \mathsf{D}(v) > c + \rho$ implies there exists an honest vertex $u \in G_p^{(t)}$ which is reachable from which v.*

Proof. Assume that $\mathsf{D}(G_p^{(t)}) - \mathsf{D}(v) > c + \rho$ but there is no honest vertex $u \in G_p^{(t)}$ which is reachable from v. Let t_v be the time at which v is generated, and let it be generated by q.

By Lemma 7, there must be a vertex u in $\mathbb{G}^{(t)}$ which is reachable from v such that $\mathsf{D}(u) \leq \mathsf{D}(\mathbb{G}^{(t)}) - c$. It must therefore be the case that u is not in $G_p^{(t)}$.

Consider that when u is generated by an honest participant at time t_u, it is immediately multicast. The only way that u is not in $G_p^{(t)}$ is if it is delayed over the network. Therefore, it must be the case that $t \leq t_u + \Delta$.

This implies:

$$\begin{aligned}
\mathsf{D}(G_p^{(t)}) - \mathsf{D}(v) &\leq \mathsf{D}(\mathbb{G}^{(t)}) - \mathsf{D}(v) \\
&= \mathsf{D}(\mathbb{G}^{(t)}) - \mathsf{D}(G_q^{(t_v)}) \\
&= \mathsf{D}(\mathbb{G}^{(t)}) - \mathsf{D}(\mathbb{G}^{(t_u)}) + \mathsf{D}(\mathbb{G}^{(t_u)}) - \mathsf{D}(\mathbb{G}^{(t_v)}) + \mathsf{D}(\mathbb{G}^{(t_v)}) - \mathsf{D}(G_q^{(t_v)}) \\
&\leq |\Psi^{(t_u,t)}| + |\Psi^{(t_v,t_u)}| + \gamma \\
&\leq 2\rho + \frac{\varepsilon}{\alpha} + \gamma \\
&= c + \rho
\end{aligned}$$

where $\mathsf{D}(\mathbb{G}^{(t_v)}) - \mathsf{D}(G_q^{(t_v)}) \leq \gamma$ by Lemma 6, $|\Psi^{(t_u,t)}| \leq \rho$ because $t \leq t_u + \Delta$ and by Definition 7, and $|\Psi^{(t_v,t_u)}| \leq \rho + \frac{\varepsilon}{\alpha}$ by an argument used in Lemma 7.

This is a contradiction with the premise of the lemma. □

Armed with Lemma 8, Lemma 3 is straightforward. We re-state it and prove it.

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

Proof. If $\mathsf{D}(G_p^{(t)}) - \mathsf{D}(v) \leq c + \rho$, this is trivial because v is a starting vertex. If $\mathsf{D}(G_p^{(t)}) - \mathsf{D}(v) > c + \rho$ then it follows from Lemma 8 that v is reachable from a starting vertex as follows. Consider the honest vertex $u \in G_p^{(t)}$ which is reachable from v by Lemma 8. If u is a starting vertex, then we are done. If not, then recursively apply Lemma 8 to u until a starting vertex is reached. The depth of the recursion is bounded by the fact that if u is reachable from v, then $\mathsf{D}(u) > \mathsf{D}(v)$. □

We now also restate and conclude the proof of Lemma 1, as it is immediate by composing Lemmas 2 and 3.

Lemma 1 (Honest Vertex Extraction). *For every time t, honest participant p active at t, and honest vertex $v \in \mathbb{G}^{(t)}$:*

$$\mathsf{D}(G_p^{(t)}) - \mathsf{D}(v) > \ell_1 \implies v \in \mathsf{extract}(G_p^{(t)})$$

Proof. This is by Lemmas 2 and 3. By Lemma 2, if $\mathsf{D}(G_p^{(t)}) - \mathsf{D}(v) > \ell_1$ then $v \in G_p^{(t)}$. By Lemma 3, if v is in $G_p^{(t)}$ then $v \in \mathsf{extract}(G_p^{(t)})$. □

B.3 Extracting Consistent Corrupt Vertices

Thus far we have shown that for any two honest participants p and q, active at t and t' respectively, for which $\mathsf{D}(G_p^{(t)}) > k + \ell_1$ and $\mathsf{D}(G_q^{(t')}) > k + \ell_1$, p and q extract the same honest vertices from their graphs up to depth k.

To complete the proof of Proposition 1, we now show an analogous consistency property of the corrupt vertices extracted by honest participants. We show that every corrupt vertex that is extracted from an honest participant's graph and is sufficiently far from the deepest vertices in the graph *must* be a predecessor of some honest vertex in the graph. We will show that consistency of extracted corrupt vertices will follow from the consistency of their honest successors. We proceed by re-stating and proving Lemma 4.

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 $\mathsf{D}(u) - \mathsf{D}(v) \leq \ell_2$.*

Sketch. We show that if a vertex v is both extracted from $G_p^{(t)}$ and is sufficiently far from the deepest vertices in $G_p^{(t)}$, then v must have an honest successor u whose depth is at most ℓ_2 more than $\mathsf{D}(v)$. Consider that if v is in $\mathsf{extract}(G_p^{(t)})$, then there must be some starting vertex z in $G_p^{(t)}$ which is reachable from v. If there is no honest vertex u reachable from v whose depth is within ℓ_2 of v, then z must be reachable from v via a long sequence of corrupt vertices which starts with v and extends either all the way to z or to some honest vertex u between v and z. Let w be the deepest corrupt vertex on this corrupt-only sequence. Intuitively, if w has an outbound edge to an honest vertex, or if w is a starting vertex in the view of any honest participant after it is generated, then the depth of w must be "close" to the depth of $\mathbb{G}^{(t_w)}$.

The proof shows that contrary to the above intuition, w is actually far from the depth of $\mathbb{G}^{(t_w)}$. We show this as follows. Because w is quite far from v (by contradiction hypothesis), there are many corrupt vertices on the path between v and w. However, if many corrupt vertices are on the path between v and w, then between t_v and t_w, many more honest vertices than corrupt vertices are generated. Those honest vertices must extend the depth of $\mathbb{G}$ so much that at t_w, w is very far (measured in depth) from the deepest vertices in $\mathbb{G}^{(t_w)}$. In fact, w is so far away from the deepest vertices in $\mathbb{G}^{(t_w)}$ that it could never be a starting vertex in the view of an honest participant, and it is not close enough to the deepest vertices in $G_{\mathcal{H}}^{(t_w)}$ (which lowerbounds the depths of honest participants at t_w) to ever gain an outbound edge to an honest vertex.

Proof. Assume for the sake of contradiction that there is a vertex v in $\mathsf{extract}(G_p^{(t)})$ such that $\mathsf{D}(G_p^{(t)}) - \mathsf{D}(v) > \ell_2$ but there is no honest vertex v' reachable from v such that $\mathsf{D}(v') - \mathsf{D}(v) \leq \ell_2$. Because v is in $\mathsf{extract}(G_p^{(t)})$ and $\mathsf{D}(G_p^{(t)}) - \mathsf{D}(v) \gg c + \rho$, there must be a starting vertex z in $G_p^{(t)}$ which is reachable from v. Moreover, there must be a path $v \to z$ in $G_p^{(t)}$.

Let w be the deepest corrupt vertex on the path $v \to z$ which is reachable from v via a path consisting of only corrupt vertices, and let t_w be the time at which w is generated. We now show that w must be quite far from v, measured in depth.

There are two cases. If there is no honest vertex on the path $v \to z$, then w is a starting vertex in $G_p^{(t)}$. Otherwise, there is an honest vertex u with an edge from w on the path $v \to z$.

(A) In the case that w is a starting vertex, it must be the case that

$$\mathsf{D}(G_p^{(t)}) - \mathsf{D}(w) \leq c + \rho \tag{6}$$

And we know by the premise of the lemma that $\mathsf{D}(G_p^{(t)}) - \mathsf{D}(v) > \ell_2$. We can therefore conclude that $\mathsf{D}(w) - \mathsf{D}(v) > \ell_2 - (c + \rho)$.

(B) In the case that there is an honest vertex u with an edge from w, there must be an honest participant q that generates u at some time $t_u > t_w$ such that

$$\mathsf{D}(G_q^{(t_u)}) - \mathsf{D}(w) \leq c \tag{7}$$

and in particular that $\mathsf{D}(u) - \mathsf{D}(w) \leq c$.
Moreover, we know by the contradiction hypothesis that $\mathsf{D}(u) - \mathsf{D}(v) > \ell_2$. We can therefore conclude that $\mathsf{D}(w) - \mathsf{D}(v) > \ell_2 - c$.

In either case, it must be true that

$$\mathsf{D}(w) - \mathsf{D}(v) > \ell_2 - (c + \rho) \tag{8}$$

Henceforth we use this relationship between w and v.

We have shown that w is quite far from v. We next we lowerbound the total number of vertices that have been generated between t_v and t_w. Then we show that during any span of time in which this many corrupt vertices have been generated, so many more honest vertices must have been generated that $\mathbb{G}$ must have grown to be much deeper than w.

Claim 8

$$|\Psi^{(t_v,t_w)}| > \frac{\frac{\ell_2-(c+\rho)}{c} - \varepsilon}{\beta} \tag{9}$$

Proof. We show the claim in two steps. We first use the distance between w and v to lowerbound the number of corrupt resources that are allocated between t_v and t_w. Then we use the number of corrupt vertices in order to lowerbound the total number of vertices which must have been generated between t_v and t_w.

Recall that w is reachable from v via a path consisting of only corrupt vertices. $\mathsf{D}(w) - \mathsf{D}(v)$ is therefore upperbounded by c times the number of vertices on the path $v \to w$, since by the protocol specification, no edge on the path may span more than c depth. Therefore,

$$c(\beta|\Psi^{(t_v,t_w)}| + \varepsilon) \geq \mathsf{D}(w) - \mathsf{D}(v) \tag{10}$$

The lowerbound on $|\Psi^{(t_v,t_w)}|$ follows from applying Inequality 8 and Inequality 10 to show

$$c(\beta|\Psi^{(t_v,t_w)}| + \varepsilon) \geq \mathsf{D}(w) - \mathsf{D}(v) > \ell_2 - (c + \rho)$$

and with algebra we arrive at Inequality 9, completing our claim. □

What remains is to show that between t_v and t_w, the depth of $\mathbb{G}$ has grown so much that for any honest participant r active at any $t' \geq t_w$, $G_r^{(t')}$ must be too deep for w to be a starting vertex in $G_r^{(t')}$ and too deep for r to add a vertex with an edge to w.

Claim 9 *For every time $t' \geq t_w$ and any honest participant r active at t', $\mathsf{D}(G_r^{(t')}) - \mathsf{D}(w) > c + \rho$.*

Proof. First, we lowerbound the difference $\mathsf{D}(G_r^{(t')}) - \mathsf{D}(w)$ in terms of $|\Psi^{(t_v,t_w)}|$. We then invoke the lowerbound on $|\Psi^{(t_v,t_w)}|$ from Claim 8 to give a concrete bound.

We start by lowerbounding $\mathsf{D}(G_r^{(t')})$ in terms of $|\Psi^{(t_v,t_w)}|$ and of $\mathsf{D}(v)$.

$$\begin{aligned}
\mathsf{D}(G_r^{(t')}) &\geq \mathsf{D}(G_{\mathcal{H}}^{(t')}) \\
&\geq \mathsf{D}(G_{\mathcal{H}}^{(t_w)}) \\
&= \mathsf{D}(G_{\mathcal{H}}^{(t_w)}) - \mathsf{D}(G_{\mathcal{H}}^{(t_v)}) + \mathsf{D}(G_{\mathcal{H}}^{(t_v)}) \\
&\geq \frac{\alpha|\Psi^{(t_v,t_w)}| - \varepsilon - \rho}{\rho} + \mathsf{D}(\mathbb{G}^{(t_v)}) - \gamma \\
&\geq \frac{\alpha|\Psi^{(t_v,t_w)}| - \varepsilon - \rho}{\rho} + \mathsf{D}(v) - \gamma
\end{aligned}$$

where $\mathsf{D}(G_{\mathcal{H}}^{(t_v)}) \geq \mathsf{D}(\mathbb{G}^{(t_v)}) - \gamma$ by a direct application of Lemma 6, and $\mathsf{D}(\mathbb{G}^{(t_v)}) \geq \mathsf{D}(v)$ trivially because $v \in \mathbb{G}^{(t_v)}$.

Recall that by Inequality 10, $\mathsf{D}(w) \leq c(\beta|\Psi^{(t_v,t_w)}| + \varepsilon) + \mathsf{D}(v)$. We can therefore lowerbound $\mathsf{D}(G_r^{(t')}) - \mathsf{D}(w)$ as a function of $|\Psi^{(t_v,t_w)}|$

$$\begin{aligned}
\mathsf{D}(G_r^{(t')}) - \mathsf{D}(w) &\geq \frac{\alpha|\Psi^{(t_v,t_w)}| - \varepsilon - \rho}{\rho} + \mathsf{D}(v) - \gamma - (c(\beta|\Psi^{(t_v,t_w)}| + \varepsilon) + \mathsf{D}(v)) \\
&= (\frac{\alpha}{\rho} - c\beta)|\Psi^{(t_v,t_w)}| - \gamma - c\varepsilon - \varepsilon - \frac{\varepsilon}{\rho} - 1
\end{aligned}$$

When plugging in our lowerbound for $|\Psi^{(t_v,t_w)}|$ from Inequality 9, we find that $\mathsf{D}(G_r^{(t')}) - \mathsf{D}(w) > c + \rho$ as claimed. □

This claim presents a contradiction with both cases above. In case (A), in which w is a starting vertex in $G_p^{(t)}$, this is a contradiction to Inequality 6. In case (B), in which w has an edge from some honest vertex u, this is a contradiction to Inequality 7. □

B.4 Consistency and Liveness of Π^G

We can now complete the proofs of Proposition 1 and Theorem 3.

Proposition 1. *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 $\mathsf{extract}(G_p^{(t)})|_k = \mathsf{extract}(G_q^{(t')})|_k$.*

Proof. Assume without loss of generality that there is a vertex v that is in $\mathsf{extract}(G_p^{(t)})|_k$ but not in $\mathsf{extract}(G_q^{(t')})|_k$. It is trivial that $\mathsf{D}(v) \leq k$ if $v \in \mathsf{extract}(G_p^{(t)})|_k$.

Assume that v is an honest vertex. By the protocol specification, v must be output by q at t' if v is extracted from $G_q^{(t')}$ because $\mathsf{D}(G_q^{(t')}) \geq \mathsf{D}(v) + \ell^*$. Therefore, v must not be extracted from $G_q^{(t')}$. But this is a contradiction with Lemma 1, which says that v must be extracted from $G_q^{(t')}$ since $\mathsf{D}(G_q^{(t')}) - \mathsf{D}(v) > \ell_1$.

Therefore, v must be a corrupt vertex. By Lemma 4, if v is in $\mathsf{extract}(G_p^{(t)})|_k$, then there must be an honest vertex u such that $\mathsf{D}(u) \leq \mathsf{D}(v) + \ell_2$ such that u is reachable from v. By Lemma 1, u must be in $\mathsf{extract}(G_q^{(t')})$ because $\mathsf{D}(G_q^{(t')}) - \mathsf{D}(u) \geq k + \ell^* - (k + \ell_2) > \ell_1$.

Because u is reachable from v, v must be in $\mathsf{extract}(G_q^{(t')})$ by the protocol specification. And because $\mathsf{D}(v) < k$ by assumption, v must be in $\mathsf{extract}(G_q^{(t')})|_k$. This is a contradiction. □

Corollary 1 (Graph Consistency). *Protocol Π^G achieves graph consistency.*

Proof. In any execution, consider any two times t, t' and p, q active at t and t', respectively. Without loss of generality, assume that $\mathsf{D}(G_p^{(t)}) \geq \mathsf{D}(G_q^{(t')})$. By Proposition 1, it must be that $\mathsf{extract}(G_p^{(t)})|_{\mathsf{D}(G_q^{(t')})-\ell^*} = \mathsf{extract}(G_q^{(t')})|_{\mathsf{D}(G_q^{(t')})-\ell^*}$, and therefore $\mathsf{extract}(G_q^{(t')})|_{\mathsf{D}(G_q^{(t')})-\ell^*} \subseteq \mathsf{extract}(G_g^{(t)})|_{\mathsf{D}(G_g^{(t)})-\ell^*}$. □

Liveness follows from the fact that an honest participant outputs every honest vertex in its local graph with depth more than ℓ^* from the end of its graph.

Lemma 9 (h-Liveness). *Protocol Π^G achieves h-liveness, for $h(N, \alpha, \varepsilon, \rho) = \alpha N - \varepsilon - \rho(\ell^* + 1)$.*

Proof. Recall that in order to compute its output, an honest participant extracts vertices from its view using the $\mathsf{extract}()$ function and then outputs the extracted vertices which are more than ℓ^* depth from the end of its graph. Recall that Lemma 3 show an honest participant always extracts every honest vertex in its local graph. We lowerbound the number of honest vertices that an honest participant outputs at any point in time by lowerbounding how many of the vertices in its view must be honest, and then upperbounding how many honest extracted vertices may have depth too high to be output.

First we lowerbound the number of vertices in an honest participant's graph which must be honest. By Claim 1, we know that at least $\alpha|\Psi^{(0,t)}| - \varepsilon - \rho$ honest vertices which have been generated from the beginning of the execution until t must be in $G_p^{(t)}$. Consider also that the total number of vertices that have been generated up to any point in time upperbounds the number of vertices in a participant's view, or $|\Psi^{(0,t)}| \geq |G_p^{(t)}.V|$. It follows that

$$|\mathsf{hon}(G_p^{(t)}.V)| \geq \alpha|\Psi^{(0,t)}| - \varepsilon - \rho \geq \alpha|G_p^{(t)}.V| - \varepsilon - \rho$$

What remains is to upperbound the number of honest vertices in a participant's graph at any point in time which are not output. Recall that an honest participant outputs all of the vertices which it extracts from its local graph up to ℓ^* depth from the end of its graph. By Claim 3, there may be at most ρ honest vertices in $\mathbb{G}^{(t)}$ with the same depth, which implies that at each depth in $G_p^{(t)}$, there may be at most ρ honest vertices. Therefore, there may be at most $\rho\ell^*$ honest vertices in G_p with depth more than $\mathsf{D}(G_p) - \ell^*$, which are therefore not output.

We conclude that $|\mathsf{extract}(G_p^{(t)})|_{\mathsf{D}(G_p^{(t)})-\ell^*}| \geq \alpha|G_p^{(t)}.V| - \varepsilon - \rho - \rho\ell^*$. The lemma follows. □

Corollary 2 (f-Liveness). *Protocol Π^G achieves f-liveness, for $f(N, \alpha, \varepsilon, \rho) = \alpha N - \varepsilon - \rho(\ell^* + 1)$.*

Proof. Immediate from Lemma 9. f-liveness is lowerbounded by h-liveness. The lowerbound is tight because an honest participant could extract no corrupt vertices from its local graph. □

C One-Bit Consensus from Graph Consensus

We now present the proof of our one-bit consensus protocol. We restate Theorem 5.

Theorem 5. *For all ρ and all ε, and for all $\alpha > \rho(1-\alpha)((3-\alpha)\rho + \frac{\varepsilon}{\alpha} + \frac{\varepsilon}{\rho} + \varepsilon + 1)$ every every (α, ε)-honest, ρ-rate-limited admissible execution of $\Pi^{\mathsf{bit}}(\alpha, \varepsilon, \rho)$ satisfies termination, agreement, and nontriviality.*

The proof of Theorem 5 follows the outline in Section 6.2. Agreement and termination are trivial, and we provide three lemmas to show nontriviality. First, we show that for every depth k in an execution, there is a time after which no (corrupt) vertex of depth k can be added to $\mathbb{G}$ which will ever be extracted by any honest participant. Second, we show the maximum number of corrupt vertices ω that can be generated from the time that $\mathbb{G}$ reaches depth k to the time when no (corrupt) vertex of depth k can ever be added and subsequently extracted by an honest participant. Finally, we show that by the time an honest participant's graph reaches depth k^*, there are more than ω honest vertices in its graph up to k^* than corrupt vertices.

Lemma 10. *Let $x = c\varepsilon + c + \rho + \frac{\varepsilon}{\rho} + 1$. For every vertex v generated at t_v: if $\mathsf{D}(G_{\mathcal{H}}^{(t_v)}) > \mathsf{D}(v) + x$, then there is no time $t \geq t_v$ and honest participant p active at t for which $v \in \mathsf{extract}(G_p^{(t)})$.*

Sketch. Recall that in order for v to be extracted from $G_p^{(t)}$, it must be either a starting vertex in $G_p^{(t)}$ or a predecessor of a starting vertex in $G_p^{(t)}$. We show that $\mathsf{D}(G_{\mathcal{H}}^{(t_v)})$ is already so much deeper than v that no honest participant which is activated in the future would ever have v as a starting vertex, and no honest participant which generates a vertex in the future would ever generate a vertex with an inbound edge from v or from any (corrupt) vertex which is reachable from v. To show this, we lowerbound the difference in depth between $G_{\mathcal{H}}$ and the deepest vertex reachable from v at any point in time $t > t_v$, as a function of the number of vertices that are generated between t and t_v. We show that the difference is always greater than $c + \rho$. Because $G_{\mathcal{H}}$ lowerbounds the view of an honest participant, we can therefore conclude that v will never be a starting vertex in any honest participant's graph and no honest participant could ever add a vertex with an inbound edge from a (corrupt) successor of v.

Proof. Assume that at the time t_v when v is generated, $\mathsf{D}(G_{\mathcal{H}}^{(t_v)}) \geq \mathsf{D}(v) + x$ and that there exists a time $t > t_v$ and honest participant p for which $v \in \mathsf{extract}(G_p^{(t)})$.

First, we claim that v cannot be a starting vertex for any honest participant's $\mathsf{extract}()$ function at any time $t' \geq t_v$. For every time $t' \geq t_v$ and every honest participant q active at t',

$$\mathsf{D}(G_q^{(t')}) \geq \mathsf{D}(G_{\mathcal{H}}^{(t')}) \geq \mathsf{D}(G_{\mathcal{H}}^{(t_v)}) > \mathsf{D}(v) + x > \mathsf{D}(v) + \rho + c$$

Therefore, because $v \in \mathsf{extract}(G_p^{(t)})$ and v is not a starting vertex in $G_p^{(t)}$, there must be a starting vertex $z \in G_p^{(t)}$ reachable from v. Specifically, if z is a starting vertex, then by definition

$$\mathsf{D}(G_p^{(t)}) - \mathsf{D}(z) < \rho + c \tag{11}$$

We now separately consider the following two cases regarding the path $v \to z$. First, we consider the case that there are no honest vertices on the path $v \to z$. Second we consider the case that there is at least one honest vertex on the path $v \to z$.

Consider the case that there are no honest vertices on the path $v \to z$. Towards contradiction with Inequality 11, we lowerbound the difference $\mathsf{D}(z) - \mathsf{D}(v)$ by upperbounding $\mathsf{D}(z)$ with respect to $\mathsf{D}(v)$ and $|\Psi^{(t_v,t)}|$, and lowerbounding $\mathsf{D}(G_p^{(t)})$ with respect to $\mathsf{D}(v)$ and $|\Psi^{(t_v,t)}|$.

First, we upperbound $\mathsf{D}(z)$. We claim that

$$\mathsf{D}(z) \leq \mathsf{D}(v) + c(\beta|\Psi^{(t_v,t)}|+\varepsilon) \tag{12}$$

By assumption, there are only corrupt vertices on the path $v \to z$. Recall from the definition of a β, ε-corrupt execution (Definition 9) that at most $\beta|\Psi^{(t_v,t)}|+\varepsilon$ corrupt vertices may be generated between t_v and t. By the protocol specification, if $v \to z$ is in an honest participant's local graph, then each edge on the path may span no more than c depth. It follows that $\mathsf{D}(z)$ is no more than $\mathsf{D}(v)$ plus c depth for every corrupt vertex generated between t_v and t.

Second, we lowerbound $\mathsf{D}(G_p^{(t)})$ using the premise of this lemma and a direct application of Lemma 5:

$$\begin{aligned}
\mathsf{D}(G_p^{(t)}) &\geq \mathsf{D}(G_{\mathcal{H}}^{(t)}) \\
&= \mathsf{D}(G_{\mathcal{H}}^{(t)}) - \mathsf{D}(G_{\mathcal{H}}^{(t_v)}) + \mathsf{D}(G_{\mathcal{H}}^{(t_v)}) \\
&\geq \frac{\alpha|\Psi^{(t_v,t)}|-\varepsilon-\rho}{\rho} + \mathsf{D}(v) + x
\end{aligned}$$

We can immediately lowerbound the difference $\mathsf{D}(G_p^{(t)}) - \mathsf{D}(z)$ using the upperbound and lowerbound just computed

$$\mathsf{D}(G_p^{(t)}) - \mathsf{D}(z) \geq (\frac{\alpha}{\rho} - c\beta)|\Psi^{(t_v,t)}|+x - c\varepsilon - \frac{\varepsilon}{\rho} - 1 \tag{13}$$

but when $\frac{\alpha}{\rho} > c\beta$ this is a contradiction with Inequality 11 because $|\Psi^{(t_v,t)}|$ must be non-negative.

Therefore, there must be an honest vertex on the path $v \to z$. Let w be the deepest corrupt vertex on the path $v \to z$ such that there are no honest vertices on the subpath $v \to w$. Then there must be an honest vertex u with an inbound edge from w. Let q be the participant that generates u, and let t_u be the time at which q generates u.

Now, because there are no honest vertices on the path $v \to w$, we can invoke the same argument that we used for the above case in which there are no honest vertices on the path $v \to z$, replacing z with w, and replacing t with the time t_u at which q generates u.

The only difference in the proof is that the difference $G_q^{(t_u)} - \mathsf{D}(w)$ is *less* than the difference $G_p^{(t)} - \mathsf{D}(z)$ above. Specifically, it must be the case that

$$\mathsf{D}(G_q^{(t_u)}) - \mathsf{D}(w) \leq c \tag{14}$$

The rest of the proof follows analogously. □

Lemma 11. *Let $\omega = \frac{\beta\rho}{\alpha}(x + \gamma + \frac{\varepsilon}{\rho} + 1) + \varepsilon$, and let the notation t_k denote the earliest time for which $\mathsf{D}(\mathbb{G}^{(t_k)}) = k$. For every time t, honest participant p active at t, and depth k: at most ω corrupt vertices in $\mathsf{extract}(G_p^{(t)})|_k$ were generated after t_k.*

Sketch We show that if more than ω corrupt vertices in $\mathsf{extract}(G_p^{(t)})|_k$ were generated after t_k, then there must be some corrupt vertex u in $\mathsf{extract}(G_p^{(t)})|_k$ which was generated when $G_{\mathcal{H}}$ was already more than x depth deeper than u. This is a contradiction to Lemma 10.

Proof. Assume that there are more than ω corrupt vertices generated between t_k and t that are in $\mathsf{extract}(G_p^{(t)})|_k$. Let u be the last such corrupt vertex that is generated, and let t_u be the time at which it is generated. Trivially, it must be the case that $t > t_u$ (otherwise u could not be in $G_p^{(t)}$).

We lowerbound $|\Psi^{(t_k,t_u)}|$ as follows. By the contradiction hypothesis, more than ω corrupt vertices have been generated between t_k and t_u. We can therefore lowerbound $|\Psi^{(t_k,t_u)}|$ using Definition 9 and the number of corrupt vertices which have been generated between t_k and t. Specifically, recall that the number of corrupt vertices that are generated between t_k and t_u is upperbounded by $\beta|\Psi^{(t_k,t_u)}|+\varepsilon$. By assumption, we have that $\beta|\Psi^{(t_k,t_u)}|+\varepsilon > \omega$, which implies that $|\Psi^{(t_k,t_u)}| > \frac{\omega-\varepsilon}{\beta}$.

Towards contradiction, we now lowerbound $\mathsf{D}(G_{\mathcal{H}}^{(t_u)})$. We do so by invoking Lemma 5 to lowerbound how much $G_{\mathcal{H}}$ must grow as honest participants add vertices to $\mathbb{G}$ between t_k and t_u. Specifically,

$$\begin{aligned}
\mathsf{D}(G_{\mathcal{H}}^{(t_u)}) &= \mathsf{D}(G_{\mathcal{H}}^{(t_u)}) - \mathsf{D}(G_{\mathcal{H}}^{(t_k)}) + \mathsf{D}(G_{\mathcal{H}}^{(t_k)}) \\
&\geq \frac{\alpha|\Psi^{(t_k,t_u)}|-\varepsilon-\rho}{\rho} + \mathsf{D}(\mathbb{G}^{(t_k)}) - \gamma \\
&> \frac{\alpha\frac{\omega-\varepsilon}{\beta}-\varepsilon-\rho}{\rho} + \mathsf{D}(\mathbb{G}^{(t_k)}) - \gamma \\
&\geq \frac{\alpha}{\beta\rho}(\omega-\varepsilon) - \gamma - \frac{\varepsilon}{\rho} - 1 + k \\
&\geq k + x
\end{aligned}$$

where it follows our definition of t_k that $\mathsf{D}(\mathbb{G}^{(t_k)}) = k$. Additionally, it follows from Lemma 6 and the definition of $G_{\mathcal{H}}^{(t_k)} = \bigcap_{t' \geq t_k, q \text{ active at } t'} G_q^{(t')}$ that $\mathsf{D}(G_{\mathcal{H}}^{(t_k)}) \geq \mathsf{D}(\mathbb{G}^{(t_k)}) - \gamma$.

This is a contradiction with Lemma 10. Recall that $\mathsf{D}(u) \leq k$ by assumption, and therefore $\mathsf{D}(G_{\mathcal{H}}^{(t_u)}) > \mathsf{D}(u)+x$. Lemma 10 says that if at the time t_u when u is generated, $\mathsf{D}(G_{\mathcal{H}}^{(t_u)}) > \mathsf{D}(u)+x$, then u may never be in $\mathsf{extract}(G_p^{(t)})$ for any p active at $t > t_u$. □

Claim 10 *For every time t and all k:* $\mathsf{D}(\mathbb{G}^{(t)}) \geq k \implies |\Psi_{\mathsf{hon}}^{(0,t)}|-|\Psi_{\mathsf{cor}}^{(0,t)}| \geq (\alpha-\beta)k - 2\varepsilon$

Proof. As a direct consequence of Definition 9, between any t and $t' > t$, $|\Psi_{\mathsf{hon}}^{(t,t')}|-|\Psi_{\mathsf{cor}}^{(t,t')}| \geq (\alpha-\beta)|\Psi^{(t,t')}|-2\varepsilon$.

Using this fact and the fact that $|\Psi^{(0,t)}| \geq \mathsf{D}(\mathbb{G}^{(t)})$ (because $\mathbb{G}^{(t)}$ can be no deeper than the number of vertices in $\mathbb{G}^{(t)}$):

$$\begin{aligned}
|\Psi_{\mathsf{hon}}^{(0,t)}|-|\Psi_{\mathsf{cor}}^{(0,t)}| &\geq (\alpha-\beta)|\Psi^{(0,t)}|-2\varepsilon \\
&\geq (\alpha-\beta)\mathsf{D}(\mathbb{G}^{(t)}) - 2\varepsilon \\
&\geq (\alpha-\beta)k - 2\varepsilon
\end{aligned}$$

□

Lemma 12. *Let $k^* = \frac{\omega+2\varepsilon}{\alpha-\beta}$. For every time t and honest participant p active at t, if $\mathsf{D}(G_p^{(t)}) \geq k^* + \ell_1$, then the majority of vertices in $\mathsf{extract}(G_p^{(t)})|_{k^*}$ are honest.*

Proof. Let t^* be the earliest time at which $\mathsf{D}(\mathbb{G}^{(t^*)}) = k^*$. We show that there are more honest vertices with depth less than k^* generated between the beginning of the execution and t^* than the sum of (a) the number of corrupt vertices generated between the beginning of the execution and t^* and (b) the total number of corrupt vertices with depth less than or equal to k^* which can be generated after t^* and still extracted from any honest participant's graph after t^*. Because all honest vertices that have been generated with depth up to k^* are guaranteed to be extracted from an honest graph with depth $k^* + \ell_1$, it follows that there must be more extracted honest vertices up to depth k^* than extracted corrupt vertices up to depth k^*.

The formal argument follows. By Claim 10, there are at least ω more honest vertices in $\mathbb{G}^{(t^*)}$ than corrupt vertices. By Lemma 2, if $\mathsf{D}(G_p^{(t)}) > k^* + \ell_1$, then all of the honest vertices generated before t^* are in $G_p^{(t)}$. By Lemma 11, there are at most ω corrupt vertices in $\mathsf{extract}(G_p^{(t)})|_{k^*}$ which were generated after t^*. Therefore, if $\mathsf{D}(G_p^{(t)}) > k^* + \ell_1$, a majority of vertices in $\mathsf{extract}(G_p^{(t)})|_{k^*}$ are honest. □

Lemma 13 (Nontriviality of Π^{bit}). *If all honest participants have input $b \in \{0,1\}$, then all honest participants that do not fail output b.*

Proof. Lemma 12 shows that for all t and participants p active at t, $\mathsf{D}(G_p^{(t)}) > k^* + \ell_1$ implies a majority of the vertices in $\mathsf{extract}(G_p^{(t)})|_{k^*}$ are honest. Therefore, if all honest participants have input b, then for all t and p active at t such that $\mathsf{D}(G_p^{(t)}) > k^* + \ell^*$, a majority of the vertices in $\mathsf{extract}(G_p^{(t)})|_{k^*}$ have label b. It is immediate that every honest participant outputs b. □