Anonymous Byzantine Consensus from Moderately-Hard Puzzles: A Model for Bitcoin

Andrew Miller
University of Central Florida
amiller@cs.ucf.edu

Joseph J. LaViola, Jr.
University of Central Florida
jjl@eecs.ucf.edu

***Abstract*—We present a formal model of synchronous processes without distinct identifiers (i.e., *anonymous processes*) that communicate using one-way public broadcasts. Our main contribution is a proof that the Bitcoin protocol achieves consensus in this model, except for a *negligible probability*, when Byzantine faults make up less than half the network. The protocol is *scalable*, since the running time and message complexity are all independent of the size of the network, instead depending only on the relative computing power of the faulty processes. We also introduce a requirement that the protocol must tolerate an arbitrary number of passive clients that receive broadcasts but can not send. This leads to a tight $2f + 1$ resilience bound.**

I. Introduction

We consider the Byzantine consensus problem in a large public peer-to-peer network: all processes in the network must come to unanimous agreement about some value, in spite of a minority of faulty processes that deviate arbitrarily from the protocol. Consensus is a powerful tool, enabling distributed applications such as dynamic name resolution [1], certificate revocation [2], mutable file systems [3], and digital currency [4]. Unfortunately, networks without admission controls are inherently vulnerable to the *Sybil* attack [5], in which selfish or malicious processes claim multiple fraudulent identities. This is detrimental to consensus protocols, which typically involve collecting a majority of votes. What stops an attacker from voting several times?

In 2009, Satoshi Nakamoto presented Bitcoin [4], a digital currency implemented as a public peer-to-peer network. At its core, Bitcoin is based on a novel Byzantine consensus protocol, in which cryptographic puzzles keep a *computationally bounded adversary* from gaining too much influence. We provide the first formal presentation of this protocol. Our model assumes synchronous processors and communications, as well as the existence of a *random oracle* [6] (i.e., an ideal hash function). We state the requirements for consensus using *Monte Carlo* rules: disagreement is permitted with positive but *negligible* probability and running time must be bounded. Our main result is a proof that Bitcoin achieves consensus when the adversary controls less than half the overall computing power (i.e., $2f + 1$ resilience). The protocol is *scalable*, in that the running time and total message bits are all independent of the size of the network, instead depending only on the ratio of faulty processes.

To make the problem more realistic, we introduce an additional requirement that even *passive clients* of the network must also decide on the same value. This requirement leads to a simple impossibility proof, since the clients must distinguish between the actual network and an impostor. Specifically, we provide a matching lower bound that shows Bitcoin achieves optimal resilience.

II. Related Work

A. Monte Carlo Consensus

Many consensus protocols use randomization, especially in order to circumvent the well-known *FLP impossibility result* [7] for deterministic asynchronous networks. The non-deterministic restatement of the consensus problem typically involves *Las Vegas* rules: the network must eventually reach consensus (with probability 1), although the amount of time taken may be unbounded [8]–[10]. King and Saia [11] suggested this requirement is too strict for large scale networks. Since deterministic consensus requires at least $O(n^2)$ message bits [12], any randomized protocol using fewer than quadratic messages must be a *Monte Carlo* protocol that fails with nonzero probability [13], [14].

We therefore introduce a global security parameter, k, and require only that a consensus protocol succeeds except for *negligible probability*, meaning a probability that approaches zero faster than the inverse of any polynomial function of k. This requirement is standard in modern cryptography [6], [15]. In particular, any instantiation of *authenticated* communications channels (as in [12]), even using idealized cryptographic primitives, will introduce an equivalent parameter.

The Bitcoin protocol meets these requirements, using a number of messages independent of the overall size of the network. If there are no faults, then the expected number of distinct messages is simply $O(k)$. In the worst case, when the number of Byzantine faults is $f = \lfloor \frac{n-1}{2} \rfloor$, the expected message complexity is $O(kn^2)$.

B. Anonymous Networks

Traditionally, the Byzantine agreement problem allows processes to be configured with distinct initial states - in particular, the mapping from unique names to processes is a known parameter, and each process knows its own name. For a large scale public network, this assumption is too strong.

Angluin [16] showed that anonymous message passing networks are strictly weaker than traditional *eponymous* networks. Other work has focused on anonymous shared-memory systems [17]–[19]. Crash-fault tolerant Consensus can be achieved in anonymous networks of unknown size [20]. Okun and Barak [21], [22] provided Byzantine agreement protocols for a stronger message-passing model that assumes authenticated channels between each pair of processes have initially been established. Delporte-Gallet et al [23] made a unifying effort in considering *homonymous* (partially anonymous) networks with a limited number of *authentic identities*. Interestingly, they found that deterministic Byzantine agreement depends only on the number of authentic identities, rather than the total number of processes. This builds the argument that anonymous (or homonymous) models are important for large networks.

C. Moderately Hard Puzzles

Aspnes suggested using *moderately-hard puzzles* to assign identities in an initially anonymous network [24]. The difficulty of the puzzles prevents a computationally bounded adversary from claiming too many identities. This is essentially the same approach taken by Bitcoin [4].

A moderately-hard computational puzzle can be solved, but only with considerable effort, time, and/or cost, as controlled by an adjustable difficulty parameter. Puzzles were first used by Merkle [25] in the development of public-key cryptography. Later, Dwork and Naor [26], [27] proposed using puzzles to slow down email spammers. Subsequently, puzzles have been used thwart Sybil attacks in a variety of settings [28]–[31]. Other uses have included timed-release encryption [32] and amortized micropayments [33].

Computational puzzles are typically constructed using a *random oracle* (i.e., an ideal hash function) that maps inputs to random outputs. The random oracle is a one-way function and can not be inverted. The puzzles are designed so that the most efficient way to find a solution is a brute force *guess-and-check* approach. Thus, the rate at which an adversary can solve puzzles is limited by the rate at which hashes can be computed.

The protocol Aspnes described [24] relies on pairwise authenticated communication channels between processes. The Bitcoin protocol eliminates this requirement by using non-interactive publicly-verifiable puzzles. The processes communicate using only unauthenticated broadcast messages.

III. Computing Model

A. Message Passing Processes

We consider a network to be a set of n identically configured processes, $\{p_1, p_2, ..., p_n\}$. An execution is a sequence of alternating states and transitions, $\pi_0, s_0, \pi_1, s_1, ..., \pi_i, s_i, ...$, where a mapping from states to sets of *available transitions* specifies the *admissible executions*. This mapping is described by the following rules:

- (*Synchronous Processors)* Each transition is associated with a monotonic timestamp, $r_i \leq r_{i+1}$, each of which is an integer multiple of a real-valued time increment dr. The distinct timestamps are called *rounds*. Each process executes its **compute** procedure exactly once per round, and each must take its turn before any process moves on to the next round. We assume the processes have identical computing power with respect to a random oracle: each process may query the oracle (i.e., call the **oracle** procedure) at most once per round. We additionally assume that a common random string, CRS, is announced at the beginning of the protocol (as in [34]). This assumption precludes an adversary that takes a head start. We will consider the *continuous limit* of this model as $dr \to 0$.
- *(Synchronous Communications)* The anonymous processes communicate using a **broadcast**(m) instruction, which places message m into mailboxes (unordered sets) associated with each process. When a message is received, the receiving process has no way of knowing where the message came from. Messages in a mailbox may be delivered in any order, and at any time, except there is a maximum delay of Δ. If m is broadcast at time r, then every process executes **receive**(m) at or before $r + \Delta$. We do not count the **receive** procedure against the total number of **oracle** queries.
- *(Nondeterminism)* The particular execution order is determined by an explicit *adversary*, a mapping from partial executions to available transitions. Processes may access a **coinflip** instruction that produces random elements in $[0, 1) \subset \mathbb{R}$. We also assume that $H : [0, 1) \to [0, 1)$ is a random oracle function that maps inputs to outputs uniformly selected in the range. The adversary has *full information* and may depend on the entire execution history including the internal state of each process (our processes have no secrets), although it may not depend on the outcomes of *future* random events. An adversary thus defines a probability distribution over the admissible executions.
- *(Byzantine faults)* The adversary is given control over f of the processes, which are called *faulty processes* or *pawns*. The adversary is *non-adaptive* and must designate its pawns at the outset. The adversary is *computationally bounded* relative to the random oracle; it may only query the oracle via its pawns, once per round per faulty process.
- *(Passive clients)* In addition to the n processes, we model an arbitrary set of *passive clients* that represents users of the network. They are identical to processes except they cannot send any messages (their **broadcast** instruction is replaced with a no-op). *Clients* means the union of passive clients and correct processes (but excluding faulty processes). Clients use an irrevocable, one-time-use instruction, **decide**(v), that outputs v as a final value.

B. Problem Statement

Each correct process p_i is initially provided with an input value, $v_i \in \{0, 1\}$. A *Monte Carlo* consensus protocol must satisfy the following three requirements:

- *(Termination)*: All clients must decide in bounded time.
- *(Agreement)*: Clients must decide on the same value (except with *negligible probability*).
- *(Unanimity)*: The decided value must be one of the inputs (with *non-negligible probability*).

Our version of the unanimity requirement prevents the adversary from influencing the network too much. In particular, if the protocol is repeated, as in a transaction processing system, then the input of a process will be chosen after an expected polynomial number of updates. Unanimity also rules out the trivial protocol that has every process decide on a fixed value.

IV. Bitcoin Consensus Protocol ($2f+1$ resilience)

The Bitcoin network [4] provides an implementation of digital money, modeled as a conserved quantity of arbitrary units. Like the stone money of Yap [35], it is unimportant if the currency itself is entirely abstract (or submerged under the sea). What really matters is that *ownership* of the currency is undisputable - everyone can agree on who owns what. Ownership in Bitcoin is determined using public key cryptography: the network unanimously agrees on an association between public keys and portions of the money supply. Money can be transferred from one public key to another using digitally signed messages. Bitcoin is therefore best understood as a distributed state machine, where the state at any given time corresponds to a ledger of account balances. We focus our attention the mechanism by which the network establishes consensus about the state for a single time instant.

In the Bitcoin consensus protocol (Algorithm 1), processes vote for their preferred values by attempting to solve a moderately hard puzzle through brute force effort (i.e., by *mining*). The particular puzzle (Algorithm 6) is based on *partial hash collision*, similar to HashCash [30]. When a correct process finds a puzzle solution, it broadcasts it to the rest of the network. Each puzzle solution is counted as a vote for a single value, and each process prefers (i.e., *mines on*), the value that appears to have the most votes.

Since no strategy for solving the puzzle does better than random guessing, the adversary is forced to compete in a race that it cannot win (similar to the consensus protocols of Chor-Israeli-Li [36] and Bracha-Rachman [37]). Our correctness proof extends and formalizes the approach suggested by Nakamoto [4], based on an analysis of Poisson processes.

Assume without loss of generality that we measure time in units equal to the message delay bound, so $\Delta = 1$. Since no strategy for finding puzzle solution outperforms querying the oracle with random inputs, we restrict our attention to a *worst-case* adversary that does precisely this. Therefore, each of the n processes performs an *i.i.d.* Bernoulli trial with success probability $p = \frac{dr}{n\mu}$. As dr approaches zero, the overall network produces puzzle solutions according to a Poisson arrival process where the interarrival time between each pair of successive solutions is an *i.i.d.* exponential variable with scale parameter μ. Let $\mathbf{A}_x$ indicate the time at which the x^{th} puzzle solution is found.

Let the random variable $\mathbf{M}_{r,i}$ be the number of votes that process p_i associates with its preferred value at time r, (i.e., $|\text{Votes}|$). Let $\mathbf{M}_r$ be the minimum of these variables among the correct processes. If $\mathbf{M}_{\hat{r}} < x$, then Line 13 of Algorithm 1 implies all clients decide on a value associated with x or more votes. If additionally $\mathbf{A}_{2x} > \hat{r}$, then fewer than $2x$ votes have

Algorithm 1: Bitcoin Consensus Protocol (for process p_i)

```
1  initially do
2      Votes ← ∅
3      Prefer ← proposed_i

4  on receive (Votes′, Prefer′) do
5      if verify (v, Prefer′)  for all  v ∈ Votes′ and
       |Votes′| > |Votes| then
6          Votes ← Votes′
7          Prefer ← Prefer′

8  on compute (r) do
9      nonce ← coinflip                // nonce ∈ [0,1) ⊂ ℝ
10     if verify (nonce, Prefer) then
11         broadcast (Votes∪{nonce}, Prefer)
13     if r ≥ r̂ then  decide Prefer
```

Algorithm 2: Bitcoin's Moderately-Hard Puzzle

```
1  Assume parameters $\mu$, $n$, and $dr$ are given. Let $p := \frac{dr}{n\mu}$.
   Also assume $CRS$ is a Common Random String
   announced at the outset of the protocol. Finally, assume
   $H : [0,1) -> [0,1)$ is a random oracle function with
   uniformly distributed random outputs.
2  procedure oracle (x)
3      return $H(x)$
4  procedure verify (Vote, Prefer)
5      hash ← oracle ($CRS$||Prefer||Vote)
6      if $\frac{\text{hash}}{\{0,1\}^{k'}} \leq p$ then return True
```

been cast in total, and hence all clients must have decided on a single unique value.

It is difficult to describe $\mathbf{M}_r$ directly since it depends on the particular adversary. Instead, we proceed with an argument by *coupling*; we construct a random process $\mathbf{B}$ related to $\mathbf{M}$ such that we can describe $\mathbf{B}$ precisely. Let B be a subset of $n - f$ correct processes. Poisson processes are infinitely divisible, so the subset B finds puzzle solutions according to a Poisson arrival process with expected interarrival time $\mu_B = \frac{\delta}{2}\mu$, where $\delta = \frac{2(n-f)}{n} > 1$ (in particular, faster than half the overall rate). We define $\mathbf{B}$ as a sequence of alternating phases: (1) *mining* and (2) *delay*. In the *mining* phase, we wait for a process in B to find a puzzle solution. When a solution is found, we switch to the *delay* phase and wait for a constant time $\Delta = 1$. Puzzle solutions found during a delay phase are disregarded. Let $\mathbf{B}_x$ denote the time at which $\mathbf{B}$ has completed x cycles through each phase. [1] We now have a useful relationship between $\mathbf{B}$ and $\mathbf{M}$:

[1] $\mathbf{A}$ could also be described as an Erlang process, $\mathbf{A}_x \sim Erlang(x, \mu)$. Process $\mathbf{B}$ is equivalent to an $M/D/1/1$ queue (in Kendall notation), where $\mathbf{B}_x$ is the time after which x clients have been served (see Chapter 9 of [38]).

Lemma IV.1. *Let $\mathbf{B}_x$ and $\mathbf{M}_r$ be defined as above. Then for any $r \geq 0$ and $x \geq 0$, $\mathbf{B}_x = r$ implies $\mathbf{M}_r \geq x$.*

Proof: Assume that the protocol begins at time $r = 0$, so $\mathbf{B}_0 = 0$ and $\mathbf{M}_0 = 0$. Suppose (for induction) that $\mathbf{B}_x = r$ and $\mathbf{M}_r \geq x$. Now, suppose $r' \geq r$ is the time at which the next puzzle solution is found by a correct process, and therefore $\mathbf{B}_{x+1} = r' + \Delta$. Since Δ is a maximum message delay, then from Lines 10-11 and 5-7 of Algorithm 1, we know that every correct process associates some value with at least $x+1$ votes by time $r' + \Delta$. Thus, $\mathbf{M}_{r'+\Delta} \geq \mathbf{M}_{r'} + 1 \geq x + 1$, and by induction the lemma holds for all r and x. □

We now construct values for the parameters μ, p, x, and $\hat{r}$, that give us $\mathbf{A}_{2x} > r > \mathbf{B}_x$ with high probability. The correct processes have a δ-advantage over half the network. We divvy up this advantage by thirds: one third each for deviation bounds on $\mathbf{A}_{2x}$ and $\mathbf{B}_x$ respectively, and one third to account for the delay phases endured by $\mathbf{B}$. Our goal is to make the puzzle difficult enough that on average, the correct processes find and propagate a solution before two solutions are found overall.

First, we define x in terms of δ and security parameter k,

$$x := k/D_{KL}(\delta^{(\frac{1}{3})} \| 1), \tag{1}$$

where $D_{KL}(\mu_P \| \mu_Q)$ is the Kullback-Leibler divergence (KL-divergence) between exponential distributions P and Q with scale parameters μ_P and μ_Q respectively. The KL-divergence is related to the number of bits needed to distinguish between two probability distributions [39]. We write out the formulas for two instances we will need later:

$$D_{KL}(\delta^{(\frac{1}{3})} \| 1) = \delta^{(\frac{1}{3})} - 1 - \log \delta^{(\frac{1}{3})} \tag{2}$$

$$D_{KL}(\delta^{-(\frac{1}{3})} \| 1) = \delta^{-(\frac{1}{3})} - 1 - \log \delta^{-(\frac{1}{3})}. \tag{3}$$

Next we solve for $\hat{r}$ such that

$$\frac{(\hat{r} - x)}{x\mu_B} = \frac{2x\mu}{\hat{r}} = \delta^{(\frac{1}{3})}, \tag{4}$$

resulting in a quadratic equation that simplifies to

$$\hat{r} := \frac{4x}{4 - \delta^{(\frac{5}{3})}}. \tag{5}$$

Next we state a lemma that these parameters are satisfactory. The proof, based on Chernoff bounds, is in the appendix.

Lemma IV.2. *Let $k > 0$ be a global security parameter, and let parameters μ, $\hat{r}$, and x be defined as in Equations 1, 4, and 5. As described above, let $\mathbf{A}$ be a Poisson arrival process with scale parameter μ; $\mathbf{A}_{2x}$ is the time of the $2x$th arrival. Let $\mathbf{B}$ be a process with alternating phases of (1) Poisson arrivals (scale parameter $\frac{\delta}{2}\mu$) followed by (2) constant time delay $\Delta = 1$; $\mathbf{B}_x$ is the time at which $\mathbf{B}$ has completed x cycles through both phases. Then $\mathbf{A}_{2x} > \hat{r} > \mathbf{B}_x$ except for a negligible probability.*

Proof: See Appendix.

Theorem IV.3. *The Bitcoin protocol (Algorithm 1) achieves consensus, except for negligible probability, in a model with anonymous synchronous processes and a minority of Byzantine faults.*

Proof: Each of the three problem requirements are met:

- *(Termination) All clients must decide in bounded time.* By Line 13 in Algorithm 1, clients decide at time $\hat{r}$.
- *(Agreement): Clients must decide on the same value (except with negligible probability).* It follows from Lemmas (IV.1) and (IV.2), with probability at least $1 - 2e^{-k}$, that all clients decide on a value with x votes, yet the network has produced fewer than $2x$ puzzle solutions in total. Therefore the value with x votes is unique.
- *(Unanimity): The decided value must be one of the inputs (with non-negligible probability).*
  Suppose we initialize a counter to 0 at the beginning of the protocol. If a correct process finds and propagates a solution before two are found in total (i.e., $\mathbf{B}_1 < \mathbf{A}_2$), then we increment the counter at time $\mathbf{B}_1$. Otherwise, we decrement the counter at time $\mathbf{A}_2$. In either case, after we modify the counter, we repeat the experiment, a total of $2x + 1$ times. The value of the counter is a simple random walk. It follows from (4) and the definitions of $\mathbf{A}$ and $\mathbf{B}$ that

  $$\mathbf{E}[\mathbf{B}_1] = \mu_B + 1 < 2\mu = \mathbf{E}[\mathbf{A}_2], \tag{6}$$

  and therefore the random walk is positive-biased. Since an odd number of steps have been taken, the final value is positive with probability better than $1/2$. If the final value is positive, then we know from the Ballot theorem [40] that with probability $\frac{1}{2x+1}$ the counter never returned to zero. When this occurs, the correct processes converge to one of their inputs in the first step and ultimately decide on that value.

V. $2f + 1$ Resilience Lower Bound

The use of *passive clients* in our model leads to a simple impossibility result based on indistinguishability. We show that $2f + 1$ is the optimal resilience for protocols using random oracles, regardless of the nature of the puzzle. If the adversary controls half the network, then it can perfectly simulate the correct processes, querying the oracle an equal number of times. Passive clients must distinguish between the correct processes and the impostors.

Theorem V.1. *In an anonymous network with passive clients, if Byzantine faulty processes make up half the network (i.e., $n \leq 2f$), then no protocol achieves consensus with high probability.*

Proof: Suppose to the contrary that a protocol exists. Divide the processes in two groups, P and Q. Consider the following two scenarios:

- *(Scenario 1):* The correct processes are in P, the pawns are in Q, and all processes are given input 0.

- *(Scenario 2):* The correct processes are in Q, the pawns are in P, and all processes are given input 1.

In either case, the pawns simulate the correct protocol as though given the opposite input. No messages are delivered between

By *unanimity*, the processes decide 0 with probability 1.

Divide the processes in two groups, P and Q. except for the wrong input. No messages are delivered between P and Q, but a passive client receives all broadcasts from both groups. The Consider two scenarios:

- *(Scenario 1):* The correct processes are in P and the pawns in Q. All processes receive input 0. The pawns simulate the correct protocol but with input 1. By *unanimity*, the passive client decides on 0 with non-negligible probability.
- *(Scenario 2):* The correct processes are in Q and the pawns in P. All processes receive input 1. The pawns simulate the correct protocol but with input 0. By *unanimity*, the correct processes agree on 0.

processes are given input 0 or the processes are given input 1; in either case, the pawns simulate the correct protocol as though they were given the opposite input. However, since the two scenarios are indistinguishable so the client can do no better than to pick a bit at random and disagree with probability $\frac{1}{2}$. $\square$

VI. Running Time and Message Complexity

We analyze the asymptotic complexity of the Bitcoin protocol. It follows from Lemma IV.2 that the number of distinct messages sent by correct processes is bounded by x, which is defined in terms of the dimensionless 'advantage' parameter δ (see Equation 1). Each message contains an entire set of votes, but most are duplicated; we only count the distinct elements. Interestingly, this bound is independent of both the total size of the network, n, and the message delay bound Δ. Since n is a number of discrete processes, and since we assume that the adversary controls fewer than half the processes, the worst tolerable parameter is $f = \lfloor \frac{n-1}{2} \rfloor$. Since $\delta = \frac{n-f}{n}$, we can calculate the limit as $n \to \infty$ and observe that the asymptotic number of messages is $x = O(n^2 k)$. Similarly, since the running time $\hat{r}$ is measured in units of Δ, we find that $\hat{r}$ is asymptotically $O(\Delta n^2 k)$.

In Section V we explained that our model requires *passive clients* to distinguish between the correct network and a computationally-challenged impostor only by observing broadcast messages. We can thus relate the Byzantine consensus problem to a *statistical test*. At each instant, every process *prefers* the value that is *most likely*, given the available information, to be preferred by the largest number of processes. For the particular puzzle we defined, the only discriminating information is the number of distinct puzzle solutions found. Consider the number of bits of certainty obtained *per message*, (i.e., $\frac{k}{x}$). From Equation 1, this is equal to the KL-divergence between two exponential distributions, where δ is the ratio between the respective scale parameters. Since the KL-divergence is a measure of the amount of information gained from a single sample from one of two distributions, our consensus protocol is information-theoretically optimal, at least for puzzles based on Bernoulli trials.

VII. Discussion

The Bitcoin consensus protocol is *scalable* in that message complexity and running time depend on the dimensionless parameter δ, rather than on the overall scale of the network, n. In this sense, it may be preferable to replace f with δ in defining the resiliency of a consensus protocol. In fact, we could state the problem more generally, using n to refer to the total computational power of the network, rather than a discrete number of processes; neither of our proofs would be affected by this change.

We would like to be able to assess the overall cost-effectiveness of Bitcoin. In a peer-to-peer network, the scale, n, is not directly controlled but instead depends on the perceived usefulness and security of the network. Instead of bounding the adversary's computational rate as a fraction of the network size (expressed in oracle queries per second), we may prefer to define an absolute bound A for the adversary's total work (expressed in oracle queries). Our proofs would be unaffected if we allowed the pawns to perform $A = f\frac{\hat{r}}{dr}$ total queries at any time, not necessarily in lockstep with the correct processes. Thus, for fixed A, an inverse relationship exists between the size of the network and the running time $\hat{r}$. To proceed further, we would also need to examine the relationship between n and A: should a larger network tolerate a proportionally stronger adversary?

It would be preferable if consensus could be achieved without needing to expend so much effort solving otherwise-useless puzzles. This apparent waste of resource may be unavoidable. We mentioned earlier that anonymous public networks are inherently vulnerable to the Sybil attack [5]. A closely related concept is the *social cost of cheap pseudonyms* [41] incurred when strangers cooperate, e.g., in internet commerce [42]. This cost must be paid, either through explicit entry fees, or else through *dues paying* as newcomers gradually build a trustworthy reputation. The Bitcoin protocol does not spare the cost of cheap pseudonyms, but rather provides a scalable and decentralized mechanism for paying it using computational resources.

The protocol we have described relies on parameters set according to correct presumptions about network characteristics, in particular the size of the network and the overall message propagation time. In an actual peer-to-peer network, these quantities can expected to change over time. Thus, without an internal mechanism to compensate, an instantiation of this network must attain consensus about these quantities through some external mechanism, begging the problem. We would prefer a consensus protocol that did not require global configuration of such parameters. We leave open the question of whether Bitcoin could be extended to function when the network size and communications delay are unknown (i.e., a protocol for the model of *partial synchrony* [43]).

VIII. Conclusion and Future Work

We presented the Bitcoin consensus protocol in a model of anonymous peer-to-peer networks. Such a protocol could enable a *world wide transaction processor*, a wide public network for which any clients will receive consistent view, regardless of their identity and their location in the network. We showed that this protocol meets its objectives, at least when knowledge of the network size and communications latency is assumed, as well as that a majority of or processes execute the protocol correctly. In future work we will attempt to improve the viability of this approach and avoid these assumptions.

References

[1] C. Cachin and A. Samar, "Secure Distributed DNS," in *Proceedings of the 2004 International Conference on Dependable Systems and Networks*, ser. DSN '04. Washington, DC, USA: IEEE Computer Society, 2004, p. 423.

[2] L. Zhou, F. B. Schneider, and R. Van Renesse, "COCA: A secure distributed online certification authority," *ACM Transactions on Computing Systems*, vol. 20, no. 4, pp. 329–368, Nov. 2002.

[3] J. Kubiatowicz, D. Bindel, Y. Chen, S. Czerwinski, P. Eaton, D. Geels, R. Gummadi, S. Rhea, H. Weatherspoon, W. Weimer, C. Wells, and B. Zhao, "Oceanstore: An architecture for global-scale persistent storage," *ACM SIGPLAN Notices*, vol. 35, no. 11, pp. 190–201, 2000.

[4] S. Nakamoto, "Bitcoin: A peer-to-peer electronic cash system," unpublished, Tech. Rep., 2009. [Online]. Available: http://bitcoin.org/bitcoin.pdf

[5] J. R. Douceur, "The sybil attack," *Peer-to-peer Systems*, vol. 2429, pp. 251–60, 2002.

[6] M. Bellare, "Random oracles are practical: A paradigm for designing efficient protocols," *Proceedings of the 1st ACM conference on*, no. November 1993, pp. 1–20, 1993.

[7] M. J. Fischer, N. a. Lynch, and M. S. Paterson, "Impossibility of distributed consensus with one faulty process," *Journal of the ACM*, vol. 32, no. 2, pp. 374–382, Apr. 1985.

[8] M. Ben-Or, "Another advantage of free choice: Completely asynchronous agreement protocols," in *Proceedings of the second annual ACM Symposium on Principles of Distributed Computing*, 1983, pp. 27–30.

[9] B. Chor and B. A. Coan, "A Simple and Efficient Randomized Byzantine Agreement Algorithm," *IEEE Transactions on Software Engineering*, vol. 11, pp. 531–539, 1985.

[10] J. Aspnes, "Randomized protocols for asynchronous consensus," *Distributed Computing*, vol. 16, no. 2-3, pp. 165–175, 2003.

[11] V. King and J. Saia, "Breaking the O(n¡sup¿2¡/sup¿) bit barrier: scalable byzantine agreement with an adaptive adversary," *Journal of the ACM (JACM)*, vol. 58, no. 4, pp. 18:1–18:24, 2011.

[12] D. Dolev and H. R. Strong, "Authenticated algorithms for Byzantine agreement," *SIAM Journal on Computing*, vol. 12, no. 4, pp. 656–666, 1983.

[13] H. Attiya and K. Censor-Hillel, "Lower bounds for randomized consensus under a weak adversary," *SIAM Journal on Computing*, vol. 39, no. 8, pp. 3885–3904, 2010.

[14] R. Tempo and H. Ishii, "Las Vegas randomized algorithms in distributed consensus problems," *European Control Conference*, vol. 13, no. 2-3, pp. 189–203, Jun. 2007.

[15] C. Cachin and V. Shoup, "Random Oracles in Constantinople: Practical Asynchronous Byzantine Agreement using," in *Proceedings of the 19th ACM Symposium on Principles of Distributed Computing*, no. July, 2000, pp. 1–26.

[16] D. Angluin, "Local and global properties in networks of processors," *Proceedings of the twelfth annual ACM symposium on Computing*, pp. 82–93, 1980.

[17] R. Guerraoui and E. Ruppert, "What Can Be Implemented Anonymously?" *Distributed Computing*, vol. 3724, pp. 244–259, 2005.

[18] Z. Bouzid, "Anonymous Agreement: The Janus Algorithm," *Principles of Distributed Systems*, vol. 7109, pp. 1–17, 2011.

[19] H. Attiya, A. Gorbach, S. Moran, and T. Technion, "Computing in Totally Anonymous Asynchronous Shared Memory," *Distributed Computing*, vol. 1499, pp. 49–61, 1998.

[20] C. Delporte-Gallet, H. Fauconnier, and A. Tielmann, "Fault-tolerant consensus in unknown and anonymous networks," in *Proceedings of the 2009 29th IEEE International Conference on Distributed Computing Systems*, 2009, pp. 368–375.

[21] M. Okun and A. Barak, "Efficient Algorithms for Anonymous Byzantine Agreement," *Theory of Computing Systems*, vol. 42, no. 2, pp. 222–238, Jul. 2007.

[22] M. Okun, "Agreement among unacquainted Byzantine generals," *Distributed Computing*, vol. 3724, pp. 499–500, 2005.

[23] C. Delporte-Gallet, "Brief announcement: Byzantine agreement with homonyms," in *Proceedings of the 22nd ACM symposium on Parallelism in algorithms and architectures*, 2010, pp. 74–75.

[24] J. Aspnes and C. Jackson, "Exposing computationally-challenged Byzantine impostors," Department of Computer Science, Yale University, New Haven, CT, Tech. Rep., 2005. [Online]. Available: ftp://cs-www.cs.yale.edu/pub/TR/tr1332.pdf

[25] R. C. Merkle, "Secure communications over insecure channels," *Communications of the ACM*, vol. 21, no. 4, pp. 294–299, Apr. 1978.

[26] C. Dwork and M. Naor, "Pricing via processing or combatting junk mail," *Advances in Cryptology - CRYPTO '92*, vol. 740, pp. 139–147, 1992.

[27] C. Dwork and A. Goldberg, "On memory-bound functions for fighting spam," *Advances in Cryptology-Crypto 2003*, pp. 1–19, 2003.

[28] A. Juels and J. Brainard, "Client Puzzles: A Cryptographic Countermeasure Against Connection Depletion Attacks," in *Proceedings of NDSS '99 (Networks and Distributed Security Systems)*, 1999, pp. 151–165.

[29] B. N. Levine, C. Shields, and N. B. Margolin, "A survey of solutions to the sybil attack," University of Massachusetts Amherst, Amherst, MA, Tech. Rep., 2006. [Online]. Available: http://prisms.cs.umass.edu/brian/pubs/levine.sybil.tr.2006.pdf

[30] A. Back, "Hashcash - A Denial of Service Counter-Measure," no. August, 2002.

[31] L. V. Ahn, M. Blum, N. J. Hopper, and J. Langford, "CAPTCHA: using hard AI problems for security," in *Proceedings of the 22nd international conference on Theory and applications of cryptographic techniques*, ser. EUROCRYPT'03, Berlin, Heidelberg, 2003, pp. 294–311.

[32] R. L. Rivest, "Time-lock puzzles and timed-release crypto," Massachusetts Institute of Technology, Cambridge, MA, Tech. Rep., 1996. [Online]. Available: http://dl.acm.org/citation.cfm?id=888615

[33] R. L. Rivest and A. Shamir, "PayWord and MicroMint : Two simple micropayment schemes," *Security Protocols*, vol. 1189, pp. 69–87, 2001.

[34] M. Blum, A. De Santis, S. Micali, and G. Persiano, "Non-Interactive Zero Knowledge," *SIAM Journal on Computing*, vol. 20, no. 6, pp. 1084–1188, 1991.

[35] C. Gillilland, "The stone money of Yap: a numismatic survey," *Smithsonian Studies in History and Technology*, vol. 23, 1975.

[36] B. Chor, A. Israeli, and M. Li, "On processor coordination using asynchronous hardware," in *Proceedings of the sixth annual ACM Symposium on Principles of distributed computing*, ser. PODC '87. New York, NY, USA: ACM, 1987, pp. 86–97.

[37] G. Bracha and O. Rachman, "Randomized consensus in expected O(n^2 log n) operations," in *Distributed Algorithms*, ser. Lecture Notes in Computer Science, S. Toueg, P. Spirakis, and L. Kirousis, Eds. Springer Berlin / Heidelberg, 1992, vol. 579, pp. 143–150.

[38] M. Mitzenmacher and E. Upfal, *Probability and computing: Randomized algorithms and probabilistic analysis*. Cambridge University Press, 2005.

[39] S. Kullback and R. A. Leibler, "On information and sufficiency," *The Annals of Mathematical Statistics*, vol. 22, no. 1, pp. 79–86, 1951.

[40] B. A. Reed, "Ballot theorems , old and new," *Horizons of Combinatorics (Bolyai Society Mathematical Studies)*, vol. 17, pp. 1–23, 2008.

[41] E. J. Friedman and P. Resnick, "The social cost of cheap pseudonyms," *Journal of Economics & Management Strategy*, vol. 10, no. 2, pp. 173–199, 2001.

[42] P. Resnick and R. Zeckhauser, "Trust among strangers in Internet transactions: Empirical analysis of eBay's reputation system," *Advances in Applied Microeconomics A Research Annual*, vol. 11, no. 12, pp. 127–157, 2002.

[43] C. Dwork, N. Lynch, and L. Stockmeyer, "Consensus in the presence of partial synchrony," *Journal of the ACM (JACM)*, vol. 35, no. 2, pp. 288–323, 1988.

APPENDIX

Recall from Section IV that we constructed two stochastic processes, $\mathbf{A}$ and $\mathbf{B}$: $\mathbf{A}$ represents the total number of puzzle solutions found by the network, while $\mathbf{B}$ is a lower bound for the minimum number of puzzle solutions that the clients associate with their preferred values. Our goal is to provide a high-probability bound that $\mathbf{A}_{2x} > \hat{r} > \mathbf{B}_x$ so all clients decide on a unique value with x votes.

Lemma A.1. *Let $k > 0$ be a global security parameter, and let parameters μ, $\hat{r}$, and x be defined as in Equations 1, 4, and 5. As described above, let $\mathbf{A}$ be a Poisson arrival process with scale parameter μ; $\mathbf{A}_{2x}$ is the time of the $2x^{th}$ arrival. Let $\mathbf{B}$ be a process with alternating phases of (1) Poisson arrivals (scale parameter $\frac{\delta}{2}\mu$) followed by (2) constant time delay $\Delta = 1$; $\mathbf{B}_x$ is the time at which $\mathbf{B}$ has completed x cycles through both phases. Then $\mathbf{A}_{2x} > \hat{r} > \mathbf{B}_x$ except for a negligible probability.*

Proof: We derive Chernoff bounds [38] that give us the desired result. Since $\mathbf{A}$ is a Poisson arrival process with an expected interarrival time μ,

$$\mathbf{A}_x \sim \sum^{x} Exp(\mu).$$

Since $\mathbf{B}$ is a process with alternating phases of a Poisson arrivals and constant delay,

$$\mathbf{B}_x \sim \sum^{x} \left(Exp(\mu_B) + \Delta\right).$$

Consider $\mathbf{A}_1$, which is an exponential distribution with scale parameter μ. The moment generating function of $\mathbf{A}_1$ is

$$\mathbf{E}[e^{t\mathbf{A}_1}] = (1 - t\mu)^{-1}. \tag{7}$$

The moment generating function for a sum of independent distributions is simply the product of the respective moment generating functions. Thus, for any x,

$$\mathbf{E}[e^{t\mathbf{A}_x}] = (1 - t\mu)^{-x}. \tag{8}$$

Process $\mathbf{B}$ is also a sum of independent variables, where the moment generating function for degenerate value $\Delta = 1$ is e^t. So,

$$\mathbf{E}[e^{t\mathbf{B}_x}] = (1 - t\mu_B)^{-x} e^{tx}. \tag{9}$$

Let $\mathbf{R}$ be a non-negative random variable. By Markov's inequality, $\Pr[\mathbf{R} \geq \hat{r}] \leq \mathbf{E}[\mathbf{R}]/\hat{r}$. This inequality provides a maximum probability of deviating from the expected value. The Chernoff bound technique is to apply this inequality to an exponential transformation on $\mathbf{R}$. For any $t > 0$, we have

$$\Pr[\mathbf{R} \geq \hat{r}] = \Pr[e^{t\mathbf{R}} \geq e^{t\hat{r}}] \leq \mathbf{E}[e^{t\mathbf{R}}]/e^{t\hat{r}}. \tag{10}$$

A similar inequality exists for $\mathbf{R} \leq \hat{r}$. For any $t < 0$, we have

$$\Pr[\mathbf{R} \leq \hat{r}] = \Pr[e^{t\mathbf{R}} \geq e^{t\hat{r}}] \leq \mathbf{E}[e^{t\mathbf{R}}]/e^{t\hat{r}}. \tag{11}$$

In order to obtain the tightest bounds, we solve for t to minimize the probability. Using this technique, and by assuming $\mathbf{E}[\mathbf{A}_{2x}] > \hat{r} > \mathbf{E}[\mathbf{B}_x]$, we obtain the following bounds:

$$\Pr[\mathbf{B}_x \geq \hat{r}] \leq \exp\left(x - (\hat{r}-x)/\mu_B - x\log\frac{x\mu_B}{(\hat{r}-x)}\right) \tag{12}$$

$$\Pr[\mathbf{A}_{2x} \leq \hat{r}] \leq \exp\left(2x - \hat{r}/\mu - 2x\log\frac{2x\mu}{\hat{r}}\right) \tag{13}$$

Beginning with (12) and applying (4) then (3), we have an bound for $\mathbf{B}$,

$$\begin{aligned}\Pr[\mathbf{B}_x \geq \hat{r}] &\leq \exp\left(x - (\hat{r}-x)/\mu_B + x\log\frac{(\hat{r}-x)}{x\mu_B}\right)\\ &= \exp\left(x\left(1 - \frac{(\hat{r}-x)}{x\mu_B} + \log\frac{(\hat{r}-x)}{x\mu_B}\right)\right)\\ &= \exp\left(-x\left(\delta^{(\frac{1}{3})} - 1 - \log\delta^{(\frac{1}{3})}\right)\right)\\ &= \exp\left(-xD_{KL}(\delta^{(\frac{1}{3})}\|1)\right)\\ &= e^{-k}.\end{aligned} \tag{14}$$

Finally, since $D_{KL}(\delta^{(\frac{1}{3})}\|1) \geq D_{KL}(\delta^{-(\frac{1}{3})}\|1)$ for all δ, we can easily provide a matching bound for $\mathbf{A}$ using (13), (4), and (2),

$$\begin{aligned}\Pr[\mathbf{A}_{2x} \leq \hat{r}] &\leq \exp\left(2x - \hat{r}/\mu + 2x\log\frac{\hat{r}}{2x\mu}\right)\\ &= \exp\left(2x\left(1 - \frac{\hat{r}}{2x\mu} - \log\frac{\hat{r}}{2x\mu}\right)\right)\\ &= \exp\left(-2x\left(\delta^{-(\frac{1}{3})} - 1 - \log\delta^{-(\frac{1}{3})}\right)\right)\\ &= \exp\left(-2xD_{KL}(\delta^{-(\frac{1}{3})}\|1)\right)\\ &\leq \exp\left(-xD_{KL}(\delta^{(\frac{1}{3})}\|1)\right)\\ &= e^{-k}.\end{aligned} \tag{15}$$

Thus, both bounds hold except for a negligible probability,

$$\Pr[\mathbf{A}_{2x} > \hat{r} > \mathbf{B}_x] \geq 1 - 2e^{-k}. \quad \square \tag{16}$$