Barracuda: The Power of ℓ-polling in Proof-of-Stake Blockchains

Giulia Fanti[†], Jiantao Jiao[§], Ashok Makkuva[‡], Sewoong Oh[‡], Ranvir Rana[‡] and Pramod Viswanath[‡]

[†]Carnegie-Mellon University, [§]University of California, Berkeley, [‡] University of Illinois at Urbana-Champaign

gfanti@andrew.cmu.edu, jiantao@eecs.berkeley.edu, {makkuva2, swoh, rbrana2, pramodv}@illinois.edu

ABSTRACT

A blockchain is a database of sequential events that is maintained by a distributed group of nodes. A key consensus problem in blockchains is that of determining the next block (data element) in the sequence. Many blockchains address this by electing a new node to propose each new block. The new block is (typically) appended to the tip of the proposer's local blockchain, and subsequently broadcast to the rest of the network. Without network delay (or adversarial behavior), this procedure would give a perfect chain, since each proposer would have the same view of the blockchain. A major challenge in practice is *forking*. Due to network delays, a proposer may not yet have the most recent block, and may therefore create a side chain that branches from the middle of the main chain. Forking reduces throughput, since only one a single main chain can survive, and all other blocks are discarded. We propose a new P2P protocol for blockchains called Barracuda, in which each proposer, prior to proposing a block, polls ℓ other nodes for their local blocktree information. Under a stochastic network model, we prove that this lightweight primitive improves throughput as if the *entire* network were a factor of ℓ faster. We provide guidelines on how to implement Barracuda in practice, guaranteeing robustness against several real-world factors.

CCS CONCEPTS

• **Mathematics of computing** → **Probabilistic algorithms**; • **Networks** → **Peer-to-peer protocols**; • **Computer systems organization** → *Dependable and fault-tolerant systems and networks*;

KEYWORDS

Stochastic networks, blockchains

1 INTRODUCTION

Blockchains are a sequential data structure in which each element depends in a structured, predefined manner on every prior element. Most blockchains implement this property recursively by including

in each data element a hash of the previous element. This makes it easy to append an element to the end of a blockchain, but difficult to alter or insert elements in the middle of a blockchain, since every subsequent element must be modified to preserve validity. In parallel, the word 'blockchain' has also come to mean the network and consensus algorithms that enable a distributed set of nodes to maintain such a data structure robustly and consistently.

In practice, there are many obstacles to maintaining a distributed blockchain, including peer churn, adversarial behavior, and unreliable networks. In this paper, we focus on the latter challenge and consider how to build efficient blockchains over unreliable networks. Although the research community is increasingly studying peer-to-peer (P2P) networks in blockchain systems [3, 4, 8, 16, 23], network effects are arguably the aspect of blockchains that have received the least attention thus far. In particular, we are interested in how the network affects blockchain performance metrics like latency and throughput for new data elements. To explain the problem, we start with a brief description of blockchain functionality.

Blockchain Primer. Blockchain systems are typically used to track sequential events, such as financial transactions in a cryptocurrency. A *block* is simply a data structure that stores a batch of such events, along with a hash of the previous block contents. The core problem in blockchain systems is determining (and agreeing on) the next block in the data structure. Many leading cryptocurrencies (e.g., Bitcoin, Ethereum, Cardano, EOS, Monero) handle this problem by electing a *proposer* who is responsible for producing a new block and sharing it with the network. This proposer election happens via a distributed, randomized protocol chosen by the system designers.

In Bitcoin, proposers are selected with probability proportional to the computational energy they have expended; this mechanism is called proof-of-work (PoW). Under PoW, each node solves a computational puzzle of random duration; upon solving the puzzle, the node relays its block over the underlying P2P network, along with proof that it solved the puzzle. Due to the high energy cost of solving PoW puzzles (or *mining*) [19], a new paradigm recently emerged called *proof-of-stake* (PoS). Under PoS, a proposer is elected with probability proportional to their stake in the system. This election process happens at fixed time intervals.

When a node is elected proposer, its job is to propose a new block, which contains a hash of the previous block's contents. Hence the proposer must choose where in the blockchain to append her new block. Most blockchains use a *longest chain* fork choice rule, under which the proposer always appends her new block to the end of the longest chain of blocks in the proposer's local view of the blocktree. If there is no network latency and no adversarial behavior, this rule ensures that the blockchain will always be a perfect chain. However, in a network with random delays, it is possible that the proposer may not have received all blocks when she is elected. As such, she

might propose a block that causes the blockchain to *fork* (e.g. Figure 2). In longest-chain blockchains, this forking is eventually resolved with probability 1 because one fork eventually overtakes the other.

Forking occurs in almost all major blockchains, and it implies that blockchains are often not chains at all, but *blocktrees*. For many consensus protocols (particularly chain-based ones like Bitcoin's), forking reduces throughput, because blocks that are not on the main chain are discarded. It also has security implications; even protocols that achieve good block throughput in the high-forking regime have thus far been prone to security vulnerabilities (which has been resolved in a recent work [2], which also guarantees low latency). Nonetheless, forking is a significant obstacle to *practical* performance in existing blockchains. There are two common approaches to mitigate forking. One is to improve the network itself, e.g. by upgrading hardware and routing. This idea has been the basis for recent projects like the Falcon network [3] and Bloxroute. The other is to design consensus algorithms that tolerate network latency by making use of forked branches. Examples include GHOST [31], SPECTRE [29], and Inclusive/Conflux [20, 21]. In this paper, we design a P2P protocol called Barracuda that effectively reduces forking for a wide class of *existing* consensus algorithms.

Contributions. We propose a novel probabilistic framework that allows one to formally investigate the trade-off between the network delays and the throughput. We propose a new block proposal protocol to mitigate the forking due to those network delays. We prove that when the proposer node polls ℓ randomly selected nodes for their local blocktree information, then it has the same effect as speeding up the communication network by a factor of ℓ, thus reducing forking significantly. This is stated informally in the following and precisely in Theorem 4.

THEOREM 1 (INFORMAL). *In a fully connected network with exponential network delays of mean Δ, let $L_\Delta(t)$ denote the (random) number of blocks included in the longest chain at time t. For sufficiently small ℓ, under the proposed ℓ-Barracuda polling, the resulting height of the longest chain is close to $L_{\Delta/\ell}(t)$ for any arbitrary block arrival process and any local attachment protocol.*

These results hold without actually changing any network hardware, and they apply generally to any block arrival process or fork choice rule. In fact, we prove a significantly stronger statement; the *entire blocktree probability mass function* changes to as if the network is faster by a factor of ℓ, not just the downstream statistic of longest chain length. The analysis also has connections to load balancing in balls-and-bins problems, which may be of independent interest. We make the following three specific contributions:

(1) We propose a new probabilistic model for the evolution of a blockchain in proof-of-stake cryptocurrencies, where the main source of randomness comes from the network delay. This captures the network delays measured in real world P2P cryptocurrency networks [8]. Simulations under this model explain the gap observed in real-world cryptocurrencies, between the achievable block throughput and the best block throughput possible in an infinite-capacity network. Our model differs from that of prior theoretical papers, which typically assume a worst-case network model that allows significant simplification in the analysis [13, 31]. We analyze

the effect of average network delay on system throughput and provide a lower bound on the block throughput.

(2) To mitigate forking due to network delays, we propose a new block proposal algorithm called ℓ-Barracuda, under which nodes poll ℓ randomly-selected nodes for their local blocktree information before proposing a new block. We show that for small values of ℓ, Barracuda has approximately the same effect as if the entire network were a factor of ℓ faster.

(3) We provide guidelines on how to implement Barracuda in practice in order to provide robustness against several real-world factors, such as network model mismatch and adversarial behavior.

Outline. We begin by describing a stochastic model for blocktree evolution in Section 2; we analyze the block throughput of this model in Section 3. Next, we present Barracuda and analyze its block throughput in Section 4. Finally, we describe real-world implementation issues in Section 5, such as how to implement polling and analyzing adversarial robustness.

2 MODEL

We propose a probabilistic model for blocktree evolution with two sources of randomness: randomness in the timing and the proposer of each new block, and the randomness in the delay in transmitting messages over the network. The whole system is parametrized by the number of nodes n, average network propagation delay Δ, proposer waiting time $\tilde{\Delta}$, and number of concurrent proposers k.

2.1 Modeling block generation

We model block generation as a discrete-time arrival process, where the tth block is generated at time $\gamma(t)$. We previously discussed the election of a single proposer for each block; in practice, some systems elect multiple proposers at once to provide robustness if one proposer fails or is adversarial. Hence at time $\gamma(t)$, k nodes are chosen uniformly at random as *proposers*, each of which proposes a distinct block. The index $t \in \mathbb{Z}^+$ is a positive integer, which we also refer to as *time* when it is clear from the context whether we are referring to t or $\gamma(t)$. The randomness in choosing the proposers is independent across time and of other sources of randomness in the model. We denote the k blocks proposed at time t as $(t, 1), (t, 2), \ldots, (t, k)$. The block arrival process follows the distribution of a certain point process, which is independent of all other randomness in the model.

Two common block arrival process are Poisson and deterministic. Under a Poisson arrival process, $\gamma(t) - \gamma(t-1) \sim \text{Exp}(\lambda)$ for some constant λ, and $\gamma(t) - \gamma(t-1)$ is independent of $\{\gamma(i)\}_{i=1}^{t-1}$. In proof-of-work (PoW) systems like Bitcoin, block arrivals are determined by independent attempts at solving a cryptographic puzzle, where each attempt has a fixed probability of success. With high probability, one proposer is elected each time a block arrival occurs (i.e., $k = 1$), and the arrival time can be modeled as a Poisson arrival process.

In many PoS protocols (e.g., Cardano, Qtum, and Particl), time is split into quantized intervals. Some protocols give each user a fixed probability of being chosen to propose the next block in each time interval, leading to a geometrically-distributed block arrival time. If the probability of selecting *any* proposer in each time slot is smaller than one, the expected inter-block arrival time will be greater than

one, as in Qtum and Particl. Other protocols explicitly designate one proposer per time slot (e.g., Cardano [6]). Assuming all nodes are active, such protocols can be modeled with a deterministic interval process, $\gamma(t) = t$, for all $t \in \mathbb{N}$. The deterministic arrival process may even be a reasonable approximation for certain parameter regimes of protocols like Qtum and Particl. If the probability of electing any proposer in a time step is close to one, there will be at least one block proposer in each time slot with high probability, which can be approximated by a deterministic arrival process. Regardless, our main results apply to arbitrary arrival processes $\gamma(t)$, including geometric and deterministic.

When a block (t, i) is generated by a proposer, the proposer attaches the new block to one of the existing blocks, which we refer to as the *parent block* of (t, i). The proposer chooses this parent block according to a pre-determined rule called a *fork-choice rule*; we discuss this further in Section 2.1. Upon creating a block, the proposer broadcasts a message containing the following information:

$$M_{t,i} = (\text{Block } (t, i), \text{pointer to the parent block of } (t, i))$$

to all the other nodes in the system. The broadcasting process is governed by our network model, which is described in Section 2.1.

In this work, we focus mainly on the PoS setting due to subtleties in the practical implementation of Barracuda (described in Section 4). In particular, PoW blockchains require candidate proposers to choose a block's contents—including the parent block—*before* generating the block. But in PoW, block generation itself takes an exponentially-distributed amount of time. Hence, if a proposer were to poll nodes before proposing, that polling information would already be (somewhat) stale by the time the block gets broadcast to the network. In contrast, PoS cryptocurrencies allow block creation to happen *after* a proposer is elected; hence polling results can be simultaneously incorporated into a block and broadcast to the network. Because of this difference, PoS cryptocurrencies benefit more from Barracuda than PoW ones.

Global view of the blocktree. Notice that the collection of all messages forms a rooted tree, called the *blocktree*. Each node represents a block, and each directed edge represents a pointer to a parent block. The root is called the *genesis block*, and is visible to all nodes. All blocks generated at time $t = 1$ point to the genesis block as a parent. The blocktree grows with each new block, since the block's parent must be an existing block in the blocktree; since each block can specify only one parent, the data structure remains a tree. Formally, we define the global blocktree as follows.

Definition 1 (Global tree). We define the *global tree* at time t, denoted as G_t, to be a graph whose edges are described by the set $\{(\text{Block } (j, i), \text{pointer to the parent block of } (j, i)) : 1 \leq j \leq t, 1 \leq i \leq k\}$ with the vertices being the union of the genesis block and all the blocks indexed as $\{(j, i) : 1 \leq j \leq t, 1 \leq i \leq k\}$.

If there is no network delay in communicating the messages, then all nodes will have the same view of the blocktree. However, due to network delays and the distributed nature of the system, a proposer might add a block before receiving all the previous blocks. Hence, the choice of the parent node depends on the local view of the blocktree at the proposer node.

Local view of the blocktree. Each node has its own local view of the blocktree, depending on which messages it has received. Upon receiving the message $M_{t,i}$, a node updates its local view as follows. If the local view contains the parent block referred in the message, then the block t is attached to it. If the local view does not contain the parent block, then the message is stored in an *orphan cache* until the parent block is received. Notice that G_t is random and each node's local view is a subgraph of G_t.

2.2 Network model and fork choice rule

We avoid modeling the topology of the underlying communication network by instead modeling the (stochastic) end-to-end delay of a message from any source to any destination node. Stochastic network models have been studied for measuring the effects of selfish mining [15] and blockchain throughput [25]. We assume each block reaches a given node with delay distributed as an independent exponential random variable with mean Δ. This exponential delay captures the varying and dynamic network effects of real blockchain networks, as empirically measured in [8] on Bitcoin's P2P network. In particular, this exponential delay encompasses both network propagation delay and processing delays caused by nodes checking message validity prior to relaying it. These checks are often used to protect against denial-of-service attacks, for instance.

When a proposer is elected to generate a new block at time $\gamma(t)$, she waits time $\tilde{\Delta} \in [0, 1)$ and decides on where to append the new block in its local blocktree. The choice of parent block is governed by the fork choice rule. The most common one is the Nakamoto protocol (longest chain), though other fork choice rules do exist. When a node is elected as a proposer under the Nakamoto protocol (or longest chain rule), the node attaches the block to the leaf of the *longest chain* in the *local* blocktree. When there is a tie, the proposer chooses one arbitrarily. Longest chain is widely-used, including in Bitcoin, ZCash, and Monero. The Nakamoto protocol belongs to the family of *local attachment protocols*, where the proposer makes the decision on where to attach the block solely based on the snapshot of its *local* tree at time $\gamma(t) + \tilde{\Delta}$, stripping away the information on the proposer of each block. In other words, we require that the protocol be invariant to the identity of the proposers of the newly generated block. We show in Section 4 that our analysis applies generally to all local attachment protocols. In practice, almost all blockchains use local attachment protocols.

Notice that if Δ is much smaller than the block inter-arrival time and all nodes obey protocol, then the global blocktree G_t is more likely to form a chain. On the other hand, if Δ is much larger than the block inter-arrival time, then G_t is more likely to be a star (i.e. a depth-one rooted tree). To maximize blockchain throughput, it is desirable to design protocols that maximize the expected length of the longest chain of G_t. Intuitively, a faster network infrastructure with a smaller Δ implies less forking. In this work, we are interested primarily in settings where Δ is larger than the mean inter-block time. This is admittedly not a conventional setting for existing blockchain systems, but a current trend in next-generation blockchains is to minimize block times and/or to run blockchains on increasingly unreliable networks (e.g., ad hoc networks, wireless networks, etc.). In both settings, we may expect Δ to be comparable

to or larger than the block time. Hence our paper aims in part to understand the feasibility of operating blockchains in this regime.

3 BLOCK THROUGHPUT ANALYSIS

A key performance metric in blockchains is *transaction throughput*, or the number of transactions that can be processed per unit time. Transaction throughput is closely related to a property called *block throughput*, also known as the main chain growth rate. Given a blocktree G_t, the length of the main chain $L(G_t)$ is defined as the number of hops from the genesis block to the farthest leaf. Precisely,

$$L(G_t) \triangleq \max_{B \in \partial(G_t)} d(B_0, B),$$

where $\partial(G_t)$ denotes the set of leaf blocks in G_t, and $d(B_0, B)$ denotes the hop distance between two vertices B_0 and B in G_t. We define *block throughput* as $\lim_{t\to\infty} \mathbb{E}[L(G_t)]/t$. Block throughput describes how quickly blocks are added to the blockchain; if each block is full and contains only valid transactions, then block throughput is proportional to transaction throughput. In practice, this is not the case, since adversarial strategies like selfish mining [10] can be used to reduce the number of valid transactions per block. Regardless, block throughput is frequently used as a stepping stone for quantifying transaction throughput [2, 13, 31].

For this reason, a key objective of our work is to quantify block throughput, both with and without polling. We begin by studying block throughput without polling under the Nakamoto protocol fork-choice rule, as in Bitcoin. This has been previously studied in [2, 13, 31], under a simple network model where there is a fixed deterministic delay between any pair of nodes. This simple network model is justified by arguing that if all transmission of messages are guaranteed to arrive within a fixed maximum delay d, then the worst case of block throughput happens when all transmission have delay of exactly d. Such practice ignores all the network effects, for the sake of tractable analysis. In this section, we focus on capturing such network effect on the block throughput. We ask the fundamental question of how block throughput depends on the average network delay, under a more realistic network model where each communication is a realization of a random exponential variable with average delay Δ. In the following (Theorem 2), we provide a lower bound on the block throughput, under the more nuanced network model from Section 2, and Nakamoto protocol fork-choice rule. This result holds for a deterministic arrival process. We refer to a longer version of this paper [11] for a proof.

THEOREM 2. *Suppose there is a single proposer ($k = 1$) at each discrete time, $\gamma(t) = t \in \{1, 2, \ldots\}$, with no waiting time ($\tilde{\Delta} = 0$). For any number of nodes n, any time t, any average delay Δ, and $C_\Delta = e^{\frac{-1}{\Delta}}$, under the Nakamoto protocol, we have that*

$$\frac{\mathbb{E}[L_{\mathsf{Chain}}(G_t)]}{t} \geq \exp\left(\frac{-C_\Delta}{(1 - C_\Delta)^2}\right).$$

Notice that trivially, $\mathbb{E}[L_{\mathsf{Chain}}(G_t)]/t \leq 1$, with equality when there is no network delay, $\Delta = 0$. Theorem 3 and our experiments in Figure 1 suggest that Theorem 2 is tight when $\Delta \ll 1$. Hence there is an (often substantial) gap between the realized block throughput and the desired upper bound. This gap is caused by network delays; since proposers may not have an up-to-date view of the blocktree due to network latency, they may append to blocks that are not
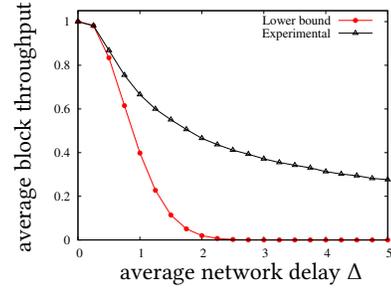

Figure 1: Block throughput vs. average network delay for an inter-block time of 1 time unit.

necessarily at the end of the global main chain, thereby causing the blockchain to *fork*.

One goal is to obtain a blocktree with no forking at all, i.e., a perfect blockchain with $L_{\mathsf{Chain}}(G_t) = t$. Setting $\exp\left(-C_\Delta/(1 - C_\Delta)^2\right) = 1 - 1/t$, which implies that $\mathbb{E}[L_{\mathsf{Chain}}(G_t)] \geq t - 1$, we obtain that $\Delta = \Theta(\frac{1}{\log t})$. The following result shows that if $\Delta = O(\frac{1}{\log t})$, then $L_{\mathsf{Chain}}(G_t) = t$ with high probability.

THEOREM 3. *Fix a confidence parameter $\delta \in (0, 1)$, $k = 1$, $\gamma(t) = t$. For the Nakamoto protocol, if*

$$\frac{1}{\Delta} \geq \frac{\left(\ln t - \ln \ln \frac{1}{\delta}\right)}{1 - \tilde{\Delta}}, \tag{1}$$

then the chain $Gen - 1 - 2 - \ldots - t$ happens with probability at least $\delta - o(1)$ as $t \to \infty$ and $n \gg t^2$.

Conversely, when $n \gg (\Delta t \ln t)^2$ and

$$\frac{1}{\Delta} \leq \frac{\left(\ln t - \ln \ln \frac{1}{\delta}\right)}{1 - \tilde{\Delta}}, \tag{2}$$

then the chain $Gen - 1 - 2 - \ldots - t$ happens with probability at most $\delta + o(1)$ as $t \to \infty$. Here $\gg$ ignores the dependence on the parameter δ, which is fixed throughout.

The proof is included in Section 6.1. This result shows the prevalence of forking. For example, if we conservatively use Bitcoin's parameters settings, taking $\Delta = 0.017$, $\tilde{\Delta} = 0$, and $\delta = 0.01$, equation (2) implies that for $t \gtrsim 5$ blocks, forking occurs with high probability. Hence forking is pervasive even in systems with parameters chosen specifically to avoid it.

A natural question is how to reduce forking, and thereby increase block throughput. To this end, we next introduce a blockchain evolution protocol called Barracuda, that effectively reduces forking without changing the system parameter Δ, which is determined by network bandwidth.

4 ℓ-BARRACUDA

To reduce forking and increase block throughput, we propose ℓ-Barracuda, which works as follows: upon arrival of a block (t, i), the proposer of block (t, i) selects $\ell - 1$ nodes in the network uniformly at random, and inquires about their local tree.[1] The proposer aggregates the information from the $\ell - 1$ other nodes and makes a

---

[1]We use the name Barracuda to refer to the general principle, and ℓ-Barracuda to refer to an instantiation with polling parameter ℓ.

decision on where to attach block (t, i) based on the *local attachment protocol* it follows. One key observation is that there is no conflict between the local trees of each node, so the Barracuda strategy simply merges totally ℓ local trees into a single tree with union of all the edges in the local trees that are polled. Note that we poll $\ell - 1$ nodes, such that a total ℓ local trees are contributing, as the proposers own local tree also contributes to the union.

We assume that when Barracuda polling happens, the polling requests arrive at the polled nodes instantaneously, and it takes the proposer node time $\tilde{\Delta}$ to make the decision on where to attach the block. The instantaneous polling assumption is relaxed in Section 5. Recall that in our model, Δ accounts for both network delay and processing delays. In live blockchain P2P networks, a substantial fraction of block propagation delays originate from the processing (e.g. validity checks) done by each node before relaying the block. These delays could grow more pronounced for blockchains with more complex processing requirements, such as smart contracts. Since these computational checks are not included in the polling process, the polling delay can be much smaller than the overall network propagation delay. To simplify the analysis, we also assume that each node processes the additional polled information in real time, but does not store the polled information. In other words, the information a node obtains from polling at time t is forgotten at time $t' > t + \tilde{\Delta}$. This modeling choice is made to simplify the analysis; it results in a lower bound on the improvements due to polling since nodes are discarding information. In practice, network delay affects polling communication as well, and we investigate experimentally these effects in Section 5.1.

To investigate the effect of polling on the blockchain, we define appropriate events on the probabilistic model of block arrival and block tree growth. We denote $X \sim \mathsf{Exp}(\lambda)$ an exponential random variable with probability density function $p_X(t) = \lambda e^{-\lambda t} \mathbb{1}(t \geq 0)$, and define set $[m] \triangleq \{1, 2, \ldots, m\}$ for any integer $m \geq 1$. For a message

$$M_{j,i} = (\text{Block } (j, i), \text{ point to the parent block of } (j, i)),$$

denote its arrival time to node m as $R_{(j,i),m}$. If m is the proposer of block (j, i), then $R_{(j,i),m} = \gamma(j) + \tilde{\Delta}$. If m is not the proposer of block (j, i), then $R_{(j,i),m} = \gamma(j) + \tilde{\Delta} + B_{(j,i),m}$, where $B_{(j,i),m} \sim \mathsf{Exp}(1/\Delta)$. It follows from our assumptions that the random variables $B_{(j,i),m}$ are mutually independent for all $1 \leq j \leq t, 1 \leq i \leq k, 1 \leq m \leq n$. We also denote the proposer of block (j, i) as $m_{(j,i)}$. To denote polled nodes, we also write $m_{(j,i)}$ as $m_{(j,i)}^{(1)}$, and denote the other $\ell - 1$ nodes polled by node $m_{(j,i)}$ as $m_{(j,i)}^{(2)}, m_{(j,i)}^{(3)}, \ldots, m_{(j,i)}^{(\ell)}$.

When block (j, i) is being proposed, we define the following random variables. Let random variable

$$e_{j,i,l,r} = \begin{cases} 1 & \text{if by the time } (j, i) \text{ was proposed, node} \\ & \quad m_{(j,i)}^{(l)} \text{ already received block } r \\ 0 & \text{otherwise} \end{cases} \quad (3)$$

Here $j \in [t], i \in [k], l \in [\ell], r \in \{(a, b) : a \in [j - 1], b \in [k]\}$. For any $r = (a, b)$, we denote $r[1] = a, r[2] = b$.

Since we will aggregate the information from the total ℓ nodes whenever a proposer proposes, we also define $e_{j,i,r} = 1 - \prod_{l=1}^{\ell}(1 - e_{j,i,l,r})$ as the event that when (j, i) was proposed, at least one node

$m_{(j,i)}^{(l)}$ has received block r. The crucial observation is that when the proposer tries to propose block (j, i), the complete information it utilizes for decision is the collection of random variables

$$\{e_{j,i,r} : r[1] \in [j - 1], r[2] \in [k]\}. \quad (4)$$

The global tree at time $\gamma(t) + \tilde{\Delta}$, denoted as G_t, is a tree consisting of $kt + 1$ blocks including the Genesis block. We are interested in the distribution of the global tree G_t. To illustrate how to compute the probability of a certain tree structure, we demonstrate the computation through an example where $k = 1, t = 3$, and $\ell = 1$.
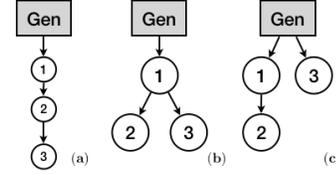

Figure 2: Examples of G_3 with varying structures.

For simplicity, we denote $e_{j,i,(r[1],r[2])}$ as $e_{j,r[1]}$ since for this example $k = 1$. The probability of some of the configurations of G_3 in Figure 2a can be written as

$$\mathbb{P}\left[G_3 = \text{Figure 2a}\right] = \mathbb{P}\left(e_{2,1} = 1, e_{3,1} = 1, e_{3,2} = 1\right),$$
$$\mathbb{P}\left[G_3 = \text{Figure 2b}\right] = \mathbb{P}\left(e_{2,1} = 1, e_{3,1} = 1, e_{3,2} = 0\right), \text{ and}$$
$$\mathbb{P}\left[G_3 = \text{Figure 2c}\right] = \mathbb{P}\left(e_{2,1} = 1, e_{3,1} = 0\right).$$

Note that for the event in Figure 2, it does not matter whether node $m_{(3,1)}$ has received block $(2, 1)$ or not, as the parent of that block is missing in $m_{(3,1)}$'s local tree. Block $(2, 1)$ is therefore not included in the local tree of node $m_{(3,1)}$ at that point in time.

4.1 Main result

Under any local attachment protocol C and any block arrival distribution, the event that $E_{C,t,g} = \{G_t = g\}$ depends on the random choices of proposers and polled nodes, $\{m_{(j,i)}^{(l)} : j \in [t], i \in [k], l \in [\ell]\}$, and the messages received at those respective nodes, $\{e_{j,i,r} : j \in [t], i \in [k], r[1] \in [j - 1], r[2] \in [k]\}$, and some additional outside randomness on the network delay and the block arrival time. The following theorem characterizes the distribution of G_t on the system parameters $t, \Delta, \ell, \tilde{\Delta}$ for a general local attachment protocol C (including the longest chain protocol). We provide a proof in Section 6.2.

THEOREM 4. *For any local attachment protocol C and any inter-block arrival distribution, define random variable $\tilde{G}_t$ which takes values in the set of all possible structures of tree G_t such that* [2]

$$\mathbb{P}(\tilde{G}_t = g) \triangleq$$
$$\mathbb{E}\left[\mathbb{1}(E_{C,t,g})\middle|\left\{m_{(j,i)}^{(l)}\right\}_{j \in [t], i \in [k], l \in [\ell]} \text{ are distinct}\right]. \quad (5)$$

We have the following results:

---

[2] The random variable $\tilde{G}_t$ is well defined, since the protocol C is assumed not to depend the identity of the proposer of each block. Hence, the conditional expectation is identical conditioned on each specific $\{m_{(j,i)}^{(l)} : j \in [t], i \in [k], l \in [\ell]\}$ whenever all $tk\ell$ nodes in it are distinct.

(a) *There exists a function F independent of all the parameters in the model such that for any possible tree structure g,*

$$\mathbb{P}(\tilde{G}_t = g) = F\left(\frac{\Delta}{\ell}, \tilde{\Delta}, g, C\right). \qquad (6)$$

(b) *The total variation distance between the distribution of G_t and $\tilde{G}_t$ is upper bounded:*

$$\mathrm{TV}\left(P_{G_t}, P_{\tilde{G}_t}\right) \leq \frac{(\ell k t)^2}{2n}. \qquad (7)$$

In the definition in Eq. (5), we condition on the event that all proposers and polled nodes are distinct. This conditioning ensures that all received blocks $e_{j,i,l,r}$'s at those nodes are independent over time j. This in turn allows us to capture the precise effect of ℓ in the main result in Eq. (6). Further, the bound in Eq. (7) implies that such conditioning is not too far from the actual evolution of the blockchains, as long as the number of nodes are large enough: $n \gg (\ell k t)^2$. While this condition may seem restrictive since $t \to \infty$, notice that in practice, many blockchains operate in *epochs* of finite duration, such that the state of the blockchain is finalized between epochs [5, 17]. Finalization means that the system chooses a single fork, and builds on the last block of that fork in the subsequent epoch. Hence, the above condition can be physically met with finite n. Moreover, in practice, n need not be so large, as we show in Figure 3. Even with $n = 10,000 < (\ell k t)^2 = 160,000$ for 4-polling, the experiments support the predictions of Theorem 4.

The main message of the above theorem is that ℓ-Barracuda effectively reduces the network delay by a factor of ℓ. For *any* local attachment protocol and *any* block arrival process, up to a total variation distance of $(\ell k t)^2/n$, the distribution of the evolution of the blocktree with ℓ-Barracuda is the same as the distribution of the evolution of the blocktree with no polling, but with a network that is ℓ times faster. We confirm this in numerical experiments (plotted in Figure 3), for a choice of $\tilde{\Delta} = 0$, $n = 10,000$, $k = 1$, $t = 100$, $\gamma(t) = t$, and the longest chain fork choice rule. In the inset we show the same results, but scaled the x-axis as Δ/ℓ. As predicted by Theorem 4, the curves converge to a single curve, and are indistinguishable from one another. We used the network model from Section 2.2.
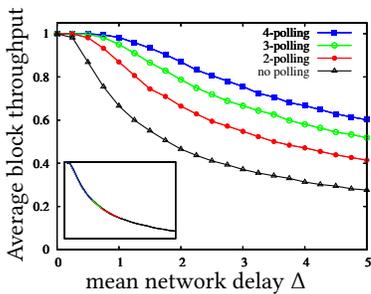

Figure 3: Comparing the average block throughput for various choices of ℓ confirms the theoretical prediction that ℓ-Barracuda effectively speeds up the network by a factor of ℓ; all curves are indistinguishable when x-axis is scaled as Δ/ℓ as shown in the inset.

Without polling, the throughput degrades quickly as the network delay increases. This becomes critical as we try to scale up PoS systems; blocks should be generated more frequently, pushing network infrastructure to its limits. With polling, we can achieve an effective speedup of the network without investing resources on hardware upgrades. Note that in this figure, we are comparing the average block throughput, which is the main property of interest. We make this connection between the throughput and ℓ precise in the following. Define $L_{\mathrm{Chain}}(G_t)$ to be the length of the longest chain in G_t excluding the Genesis block. Throughput is defined as $\mathbb{E}[L_{\mathrm{Chain}}(G_t)]/t$. We have the following Corollary of Theorem 4.

COROLLARY 1. *There exists a function $L(\Delta/\ell, \tilde{\Delta}, C)$ independent of all the parameters in the model such that*

$$\left| \mathbb{E}[L_{\mathrm{Chain}}(G_t)] - \mathbb{E}\left[L\left(\frac{\Delta}{\ell}, \tilde{\Delta}, C\right)\right] \right| \leq \frac{t(\ell k t)^2}{2n}. \qquad (8)$$

In other words, in the regime that $n \gg t^3(k\ell)^2$, the expectation of the length of the longest chain depends on the delay parameter Δ and the polling parameter ℓ only through their ratio Δ/ℓ. Hence, the block throughput enjoys the same polling gain, as the distribution of the resulting block trees.

4.2 Connections to balls-in-bins example

In this section, we give a brief explanation of the balls-in-bins problem and the power of two choices in load balancing. We then make a concrete connection between the blockchain problem and the power of ℓ-polling in information balancing.

In the classical balls-in-bins example, we have t balls and t bins, and we sequentially throw each ball into a uniformly randomly selected bin. Then, the maximum loaded bin has load (i.e. number of balls in that bin) scaling as $\Theta(\log t/\log\log t)$ [24]. The result of *power of two choices* states that if every time we select ℓ ($\ell \geq 2$) bins uniformly at random and throw the ball into the *least* loaded bin, the maximum load enjoys an near-*exponential* reduction to $\Theta(\log\log t/\log \ell)$ [24].

Our polling idea is inspired by this power of two choices in load balancing. We make this connection gradually more concrete in the following. First, consider the case when the underlying network is extremely slow such that no broadcast of the blocks is received. When there is no polling, each node is only aware of its local blockchain consisting of only those blocks it generated. There is a one-to-one correspondence to the balls-in-bins setting, as blocks (balls) arriving at each node (bin) build up a load (local blockchain). When there are t nodes and t blocks, then it trivially follows that the length of the longest chain scales as $\Theta(\log t/\log\log t)$, when there is no polling.

The main departure is that in blockchains, the goal is to maximize the length of the longest chain (maximum load). This leads to the following fundamental question in the balls-in-bins problem, which has not been solved, to the best of our knowledge. If we throw the ball into the *most* loaded bin among ℓ randomly chosen bins at each step, how does the maximum load scale with t and ℓ? That is, if one wanted to maximize the maximum load, leading to load unbalancing, how much gain does the power of ℓ choices give? We give a precise answer in the following.

THEOREM 5. *Given t empty bins and t balls, we sequentially allocate balls to bins as follows. For each ball, we select uniformly at random ℓ bins, and put the ball into the maximally-loaded bin among the ℓ chosen ones. Then, the maximum load of the t bins after the placement of all t balls is at most*

$$C \cdot \ell \cdot \frac{\log t}{\log \log t} \tag{9}$$

with probability at least $1 - \frac{1}{t}$, where $C > 0$ is a universal constant.

We refer to a longer version of this paper [11] for a proof. This shows that the gain of ℓ-polling in maximizing the maximum load is linear in ℓ. Even though this is not as dramatic as the exponential gain of the load balancing case, this gives a precise characterization of the gain in the throughput of ℓ-Barracuda in blockchains when $\Delta \gg 1$. This is under a slightly modified protocol where the polling happens in a bidirectional manner, such that the local tree and the newly appended block of the proposer are also sent to the polled nodes.

For moderate to small Δ regime, which is the operating regime of real systems, blocktree evolution is connected to a generalization of the balls-in-bins model. Now, it is as if the balls are copied and broadcasted to all other bins over a communication network. This is where the intuitive connection to balls-and-bins stops, as we are storing the information in a specific data structure that we call blocktrees. However, we borrow the terminology from 'load balancing', and refer to the effect of polling as 'information balancing', even though load balancing refers to *minimizing* the maximum load, whereas information balancing refers to *maximizing* the maximum load (longest chain) by balancing the information throughout the nodes using polling.

5 SYSTEM AND IMPLEMENTATION ISSUES

We empirically verify the robustness of our proposed protocol under various issues that might come up in a practical implementation of ℓ-Barracuda. Our experiment consists of n nodes connected via a network which emulates the end to end delay as an exponential distribution; this model is inspired by the measurements of the Bitcoin P2P network made in [8].

Each of the n nodes maintains a local blocktree which is a subset of the global blocktree. We use a deterministic block arrival process with $\gamma(t) = t$, i.e. we assume a unit block arrival time which is also termed as an epoch in this section. This represents an upper bound on block arrivals in real-world PoS systems, where blocks can only arrive at fixed time intervals. At the start of arrival t, k proposers are chosen at random and each of these proposers proposes a block.

When there is no polling, each proposer chooses the most eligible block from its blocktree to be a parent to the block it is proposing, based on the fork choice rule. In the case of ℓ-Barracuda, the proposer sends a pull message to $\ell - 1$ randomly chosen nodes, and these nodes send their block tree back to the proposer. The proposer receives the block trees from the polled nodes after a delay $\tilde{\Delta}$, and updates her local blocktree by taking the union of all received blocktrees. The same fork choice rule is applied to decide the parent to the newly generated block. In all experiments, Nakamoto longest chain fork choice rule is used. Experiments are run for $T = 100$ time epochs on a network with $n = 10,000$ nodes with $k = 1$.

5.1 Effect of polling delay

In reality, there is delay between initializing a poll request and receiving the blocktree information. We expect polling delay to be smaller than the delay of the P2P relay network because polling communication is point-to-point rather than occurring through the P2P relay network. To understand the effects of polling delay, we ran simulations in which a proposer polls $\ell - 1$ nodes at the time of proposal, and each piece of polled information arrives after time $\tilde{\Delta}_1, \tilde{\Delta}_2, .., \tilde{\Delta}_{\ell-1} \sim \text{Exp}(\frac{1}{0.1\Delta})$. The proposer determines the pointer of the new block when all polled messages are received.

Figure 5 shows the effect of such polling delay, as measured by $\Delta_{0.8}(\ell)$, the largest delay Δ that achieves a block throughput of at least 0.8 under ℓ-Barracuda. More precisely,

$$\Delta_{0.8}(\ell) = \max \left\{ \Delta \; : \; \lim_{t \to \infty} \frac{\mathbb{E}\left[L(G_t)\right]}{t} \geq 0.8 \right\}.$$

Under this model, polling more nodes means waiting for more responses; the gains of polling hence saturate for large enough ℓ, and there is an appropriate practical choice of ℓ that depends on the interplay between the P2P network speed and the polling delay.

In practice, there is a strategy to get a large polling gain, even with delays: the proposer polls a large number of nodes, but only waits a fixed amount of time before making a decision. Under this protocol, polling more nodes can only help; the only cost of polling is the communication cost. The results of our experiments under this protocol are illustrated in Figure 5 ('poll delay fixed wait' curve).

This implies a gap in our model, which does not fully account for the practical cost of polling. In order to account for polling costs, we make the model more realistic by assigning a small and constant delay of 0.01Δ to set up a connection with a polling node, and assume that the connection setup occurs sequentially for $\ell - 1$ nodes. The proposer follows the same strategy as above: waiting for a fixed amount of time before making the decision. We see that under such model, there is a finite optimal ℓ as shown in Figure 6.
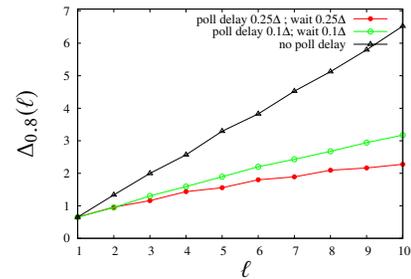

Figure 4: The polling gain continues for large ℓ even with a more practical choice of a polling delay 0.25Δ.

As practical polling delays might be larger than $0.1D$, we compare it to a more practical setting where polling delay is $D/4$ with a threshold wait time of $D/4$ in Figure 4. With this larger delays, the performance is still continuously increasing with ℓ, and still provides 250% improvement at ℓ=10.
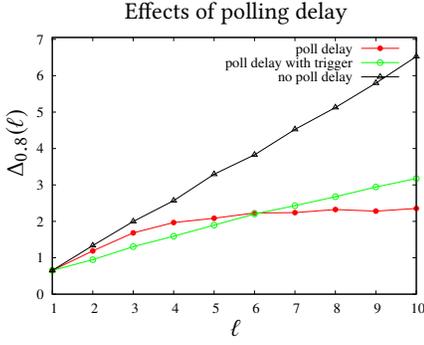
Figure 5: With a polling delay, the performance saturates after $\ell = 6$. However, we can continuously harness polling gain if the proposers propose a new block after a fixed time without waiting for all polling to arrive.
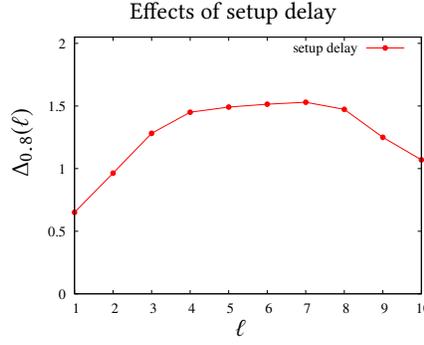
Figure 6: We assume a polling delay of $\text{Exp}(1/(0.1\Delta))$ but the proposer waits exactly $\tilde{\Delta} = 0.1\Delta$ time before proposing. When there is a setup delay $\propto \ell$, we see an optimal ℓ, which depends on all system parameters.
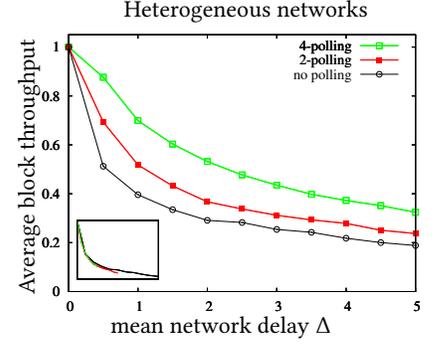
Figure 7: Heterogeneous networks enjoy the same polling gain as homogeneous ones. Heterogeneous ℓ-Barracuda provides a speedup of the network by a factor of about ℓ, as shown in the inset where the x-axis is scaled by Δ/ℓ.

5.2 Heterogeneous networks

The theoretical and experimental evidence of the benefits of ℓ-Barracuda have so far been demonstrated in the context of a homogeneous network: all the nodes in the network have the same bandwidth and processing speeds. Further, individual variation in end-to-end delay due to network traffic is captured by statistically-identical exponential random variables. In practice, heterogeneity is natural: some nodes have stronger network capabilities. We model this by clustering the nodes into h different groups based on average network speed. The speed of a connection is determined by the speed of the slower node. We compare the performance of ℓ-Barracuda with no polling (which has worse performance and serves as a lower bound). We follow the following uniform polling strategy: Let the delay Δ of a node be a part of the set $\mathcal{D} = \{\Delta_1, \Delta_2, .., \Delta_h\}$; a node's delay is defined as follows: the average delay of transmitting a block across the P2P network from node with delay Δ_i to a node with delay Δ_j is $\max(\Delta_i, \Delta_j)$ $\forall i \in [h]$. In Figure 7, we show the performance of a heterogeneous network with $h = 2$: half of the nodes have delay Δ and the others have delay 5Δ. Every node has the same proposer election probability. ℓ-Barracuda gives a throughput increase in line Theorem 4.

5.3 Other practical issues

There are remaining three major practical issues. First, the polling studied in this paper requires syncing of the complete local block-tree, which is redundant and unnecessarily wastes network resources. For efficient bandwidth usage, we propose (ℓ, b)-polling, where the polled nodes only send the blocks that were generated between times $t - 1$ and $t - b$. Secondly, to ensure timely response from polled nodes, we propose appropriate incentive mechanism, motivated by the reputation systems used in BitTorrent. Finally, a fraction of the participants may deviate from the proposed protocol with explicit malicious intent (of causing harm to the key performance metrics). It is natural to explore potential security vulnerabilities exposed by the ℓ-Barracuda protocol proposed in this paper. All these practical issues are expanded in detail with

numerical experiments to support them, in the longer version of this paper available at [11].

6 PROOFS OF THE MAIN RESULTS

6.1 Proof of Theorem 3

We apply Theorem 4 with general $\ell \geq 1$ and then specialize it to $\ell = 1$ to obtain the theorem statement. Denote the chain as g, and $e_{j,i,r[1],r[2]}$ as $e_{j,r[1]}$ since here $k = 1$. The event $E_{C,t,g}$ can be written as

$$E_{C,t,g} = \mathbb{1}(e_{2,1} = 1) \cdot \mathbb{1}(e_{3,1} = 1, e_{3,2} = 1) \cdot$$
$$\ldots \cdot \mathbb{1}(e_{t,1} = 1, e_{t,2} = 1, \ldots, e_{t,t-1} = 1). \quad (10)$$

Let $\tilde{E}$ denote the event that every node has proposed or been polled at most once. Conditioned on $\tilde{E}$, and defining $\alpha \triangleq e^{\tilde{\Delta}l/\Delta}$:

$$\mathbb{E}[\mathbb{1}(E_{C,t,g})|\tilde{E}] = \prod_{j=2}^{t} \mathbb{E}[\mathbb{1}(e_{j,1} = 1, e_{j,2} = 1, \ldots, e_{j,j-1} = 1)|\tilde{E}]$$

$$= \prod_{j=2}^{t}\prod_{m=1}^{j-1}(1 - e^{\frac{\tilde{\Delta}l}{\Delta}}e^{-\frac{m\ell}{\Delta}}) = \prod_{j=1}^{t-1}(1 - \alpha e^{-\frac{j\ell}{\Delta}})^{t-j} \leq (1 - \alpha e^{-\frac{\ell}{\Delta}})^{t-1}.$$

We now claim that if $\ell \geq \frac{\Delta(\ln t - \ln \ln \frac{1}{\delta})}{1 - \tilde{\Delta}}$, we have $\mathbb{E}[\mathbb{1}(E_{C,t,g})|\tilde{E}] \geq \delta - o(1)$. Let $c = \ln \frac{1}{\delta}$. Indeed, in this case, we have $\alpha e^{-\ell/\Delta} \leq \ln(1/\delta)/t$. Hence,

$$\mathbb{E}[\mathbb{1}(E_{C,t,g})|\tilde{E}] \geq \prod_{j=1}^{t-1}\left(1 - \frac{c^j}{t^j}\right)^{t-j}$$

$$= \left(1 - \frac{c}{t}\right)^{t\frac{t-1}{t}}\prod_{j=2}^{t-1}\left(1 - \frac{c^j}{t^j}\right)^{t-j} \overset{(a)}{\geq} e^{-c} - o(1) = \delta - o(1),$$

where (a) follows from Lemma 1 and the fact that $\lim_{t\to\infty}(1 - c/t)^{t-1} = e^{-c}$. Conversely, we show that if $\ell \leq \frac{\Delta(\ln t - \ln \ln \frac{1}{\delta})}{1 - \tilde{\Delta}}$, then $\mathbb{E}[\mathbb{1}(E_{C,t,g})|\tilde{E}] \leq \delta + o(1)$. Indeed, in this case we have $\alpha e^{-\ell/\Delta} \geq$

$\ln(1/\delta)/t$, and

$$\mathbb{E}[\mathbb{1}(E_{C,t,g})|\tilde{E}] \leq (1 - c/t)^{t-1} = e^{-c} + o(1) = \delta + o(1).$$

LEMMA 1. *Let $c > 0$ be fixed. Then we have that* $\lim_{n\to\infty} \sum_{k=2}^{n-1}(n-k)\log(1 - c^k/n^k) = 0$.

The proof is included in the extended version [11]. Note that the distribution of $\{m_1, m_2, m_3, \ldots, m_t\}$ is independent of $\{R_{ti} : t \geq 1, i \in [n]\}$, hence we could condition on a specific realization of $\{m_2, m_3, \ldots, m_T\}$ and compute the conditional expectation of the event that leads to the final global tree as a chain. We claim:

LEMMA 2. *For any $T \geq 1$, letting $x = e^{-\lambda}$, we have* $\prod_{j=1}^{T-1}(1 - x^j)^{T-j} \leq \mathbb{E}[E_T|\{m_i : 2 \leq i \leq T\}] \leq (1 - x)^{T-1}$.

The full proof is included in [11]. The upper bound uses the independence of the propagation delays, whereas the lower bound relies on the fact that all the $T - 1$ events in the $T - 1$ indicators functions of E_T are nonnegatively correlated.

Since Lemma 2 does not depend on the values of $\{m_i\}_{i=2}^{T}$, we know that the bounds apply to $\mathbb{E}[E_T]$ as well. For the d-polling strategy, it can be verified that both the upper and lower bound computations in Lemma 2 are still valid, if we replace x with x^d. Indeed, for the upper bound, each block contributes at least d independent random variables; for the lower bound, we can show that the $T - 1$ events are positively correlated. Now we claim that in order to ensure that $\mathbb{E}[E_T] \geq \delta$, the required number of d is at least approximately $d \geq \frac{\ln T - \ln \ln \frac{1}{\delta}}{\lambda}$ for T large. Let $c = \ln \frac{1}{\delta}$. We claim that if $x \leq \frac{c}{T}$, then the probability lower bound is satisfied. Indeed, in this case,

$$\mathbb{E}[E_T] \geq \prod_{j=1}^{T-1}\left(1 - \frac{c^j}{T^j}\right)^{T-j} = \left(1 - \frac{c}{T}\right)^{T \cdot \frac{T-1}{T}} \prod_{j=2}^{T-1}\left(1 - \frac{c^j}{T^j}\right)^{\frac{T^j}{c^j} \cdot \frac{c^j(T-j)}{T^j}}$$
$$\geq e^{-c} - o(1) = \delta - o(1)$$

as $T \to \infty$. We also claim that if $x > \frac{c}{T}$, then the probability lower bound is asymptotically not satisfied. Indeed, in this case

$$\mathbb{E}[E_T] \leq (1 - x)^{T-1} < \left(1 - \frac{c}{T}\right)^{T-1} = e^{-c} + o(1) = \delta + o(1)$$

as $T \to \infty$. Hence, the threshold we aim for should be precisely $x = \frac{c}{T}$. Replacing it with $e^{-\lambda d} = \frac{\ln \frac{1}{\delta}}{T}$, we get $d = \frac{\ln T - \ln \ln \frac{1}{\delta}}{\lambda}$.

6.2 Proof of Theorem 4

Part (1). One key observation is that, if every node has only been polled or proposed at most once, i.e., the set $\{m_{(j,i)}^{(l)} : j \in [t], i \in [k], l \in [\ell]\}$ contains $tk\ell$ distinct nodes, then conditioned on this specific sequence $\{m_{(j,i)}^{(l)} : j \in [t], i \in [k], l \in [\ell]\}$, all the random variables $\{e_{j,i,l,r} : j \in [t], i \in [k], l \in [\ell], r[1] \in [j-1], r[2] \in [k]\}$ are mutually independent. Furthermore, conditioned on this specific sequence, we have

$$\mathbb{E}[e_{j,i,l,r}|\{m_{(j,i)}^{(l)} : j \in [t], i \in [k], l \in [\ell]\}, \{\gamma(i)\}_{i=1}^{t}] \quad (11)$$

$$= 1 - e^{-(\gamma(j)-\gamma(r[1])-\tilde{\Delta})/\Delta}, \quad (12)$$

for all r such that $r[1] \in [j-1], r[2] \in [k]$. Let $\tilde{E}$ denote the event that $\{m_{(j,i)}^{(l)} : j \in [t], i \in [k], l \in [\ell]\}$ are distinct. It follows from the definition of local attachment protocol C that $\mathbb{E}[e_{j,i,l,r}|\tilde{E}, \{\gamma(i)\}_{i=1}^{t}] = 1 - e^{-(\gamma(j)-\gamma(r[1])-\tilde{\Delta})/\Delta}$ for all r such that $r[1] \in [j-1], r[2] \in [k]$. Note that the event $E_{C,t,g} = \{G_t = g\}$ only depends on $\{m_{(j,i)}^{(l)} : j \in [t], i \in [k], l \in [\ell]\}$ and $\{e_{j,i,r} : j \in [t], i \in [k], r[1] \in [j-1], r[2] \in [k]\}$ plus some additional outside randomness. Since $e_{j,i,r} = 1 \Leftrightarrow \sum_{l \in [\ell]} e_{j,i,l,r} \geq 1$, it follows from the independence of $e_{j,i,l,r}$ and equation (11) that

$$\mathbb{E}[e_{j,i,r}|\tilde{E}, \{\gamma(i)\}_{i=1}^{t}] = 1 - e^{-(\gamma(j)-\gamma(r[1])-\tilde{\Delta})\ell/\Delta} \quad (13)$$

all r such that $r[1] \in [j-1], r[2] \in [k]$. Hence, we have

$$\mathbb{P}(\tilde{G}_t = g) = \mathbb{E}\left[\mathbb{1}(E_{C,t,g})|\tilde{E}, \{\gamma(i)\}_{i=1}^{t}\right] = F(\{\gamma(i)\}_{i=1}^{t}, \frac{\Delta}{\ell}, \tilde{\Delta}, g, C).$$

Now we show the second part of Theorem 4. Denote by $A = \{g_1, g_2, \ldots, g_A\}$ any collection of distinct tree structures that G_t may take values in. Then, we have

$$\mathbb{P}(G_t \in A|\{\gamma(i)\}_{i=1}^{t}) = \mathbb{E}\left[\sum_{i=1}^{A} \mathbb{1}(E_{C,t,g_i})\bigg|\{\gamma(i)\}_{i=1}^{t}\right] \quad (14)$$

$$= \mathbb{P}(\tilde{G}_t \in A) + (1 - \mathbb{P}(\tilde{E}|\{\gamma(i)\}_{i=1}^{t}))$$

$$\times \left(\mathbb{E}[\sum_{i=1}^{A} \mathbb{1}(E_{C,t,g_i})|\tilde{E}^c, \{\gamma(i)\}_{i=1}^{t}] - \mathbb{E}[\sum_{i=1}^{A} \mathbb{1}(E_{C,t,g_i})|\tilde{E}, \{\gamma(i)\}_{i=1}^{t}]\right).$$
$$(15)$$

It follows from the birthday paradox computation [24, Pg. 92] that $1 - \mathbb{P}(\tilde{E}|\{\gamma(i)\}_{i=1}^{t}) \leq kt\ell(kt\ell - 1)/2n$. Hence, we have shown that for any measurable set A that G_t or $\tilde{G}_t$ take values in, we have $|\mathbb{P}(G_t \in A|\{\gamma(i)\}_{i=1}^{t}) - \mathbb{P}(\tilde{G}_t \in A)| \leq 1 - \mathbb{P}(\tilde{E}|\{\gamma(i)\}_{i=1}^{t}) \leq \frac{kt\ell(kt\ell-1)}{2n}$. The result follows from the definition of the total variation distance $\mathrm{TV}(P_{G_t|\{\gamma(i)\}_{i=1}^{t}}, P_{\tilde{G}_t}) = \sup_A |\mathbb{P}(G_t \in A|\{\gamma(i)\}_{i=1}^{t}) - \mathbb{P}(\tilde{G}_t \in A)|$.

Part (2). We note that there exists some function $L(\{\gamma(i)\}_{i=1}^{t}, \frac{\Delta}{\ell}, \tilde{\Delta}, C)$ independent of all the parameters in the model such that the expectation of the longest chain of $\tilde{G}_t$ is equal to $L(\{\gamma(i)\}_{i=1}^{t}, \frac{\Delta}{\ell}, \tilde{\Delta}, C)$. To obtain the final result, it suffices to use the variational representation of total variation distance, $\mathrm{TV}(P, Q) = \sup_{f:|f| \leq \frac{1}{2}} \mathbb{E}_P f - \mathbb{E}_Q f$, and taking $f = \frac{1}{t} \cdot (L_{\mathrm{Chain}}(G_t) - t/2)$, upon noticing that the length of the longest chain in the tree G_t is at most t.

7 RELATED WORK

Four main approaches exist for reducing forking.

(1) Reducing proposer diversity. A natural approach is to make the same node propose consecutive blocks; for instance, Bitcoin-NG [9] proposers use the longest-chain fork choice rule, but within a given time epoch, only a single proposer can propose blocks. This allows the proposer to quickly produce blocks without forking effects. Although Bitcoin-NG has high throughput, it exhibits a few problems. When a single node is in charge of block proposal for an extended period of time, attackers may be able to learn that node's IP address and take it down. The idea of fixing the proposer is also used in other protocols, such as Thunderella [27] and ByzCoin [18].

(2) Embracing forking. Other protocols use forking to contribute to throughput. Examples include GHOST [31], PHANTOM [30],

SPECTRE [29], and Inclusive/Conflux [20, 21]. GHOST describes a fork choice rule that tolerates honest forking by building on the heaviest subtree of the blocktree. SPECTRE, PHANTOM, and Conflux instead use existing fork choice rules, but build a directed acyclic graph (DAG) over the produced blocks to define a transaction ordering. A formal understanding of such DAG-based protocols is evolving; their security properties are not yet well-understood.

(3) Structured DAGs. A related approach is to allow *structured* forking. The Prism consensus mechanism explicitly co-designs a consensus protocol and fork choice rule to securely deal with concurrent blocks, thereby achieving optimal throughput and latency [2]. The key intuition is to run many concurrent blocktrees, where a single proposer tree is in charge of ordering transactions, and the remaining voter trees are in charge of confirming blocks in the proposer tree. Barracuda is designed to be integrated into *existing* consensus protocols, whereas [2] is a new consensus protocol.

(4) Fork-free consensus. Consensus protocols like Algorand [14], Ripple, and Stellar [22] prevent forking entirely by conducting a full round of consensus for every block. Although voting-based consensus protocols consume additional time for each block, they may improve overall efficiency by removing the need to resolve forks later; this hypothesis remains untested. A challenge in such protocols is that BFT voting protocols can be communication-intensive, and require a known set of participants. Although some work addresses these challenges [18, 26], many industrial blockchain systems running on BFT voting protocols require some centralization.

Our approach can be viewed as a partial execution of a polling-based consensus protocol. Polling has long been used in consensus protocols [1, 7, 12, 28]. Our approach differs in part because we do not use polling to reach complete consensus, but to reduce the number of inputs to a (separate) consensus protocol.

8 CONCLUSION

In this paper, we propose ℓ-polling as a technique for improving block throughput in proof-of-stake cryptocurrencies. We show that for small ℓ, ℓ-polling has the same effect on block throughput as if the mean network delay were reduced by a factor of ℓ. This simple, lightweight method improves throughput without substantially altering either the underlying consensus protocol or the network. Several open questions remain, particularly with regards to analyzing adversarial behavior in ℓ-polling. We have avoided them in this paper by proposing a symmetric version of the protocol (cf. Section 5.3), but even within the original ℓ-polling protocol, it is unclear how much an adversary could affect block throughput and/or chain quality by responding untruthfully to poll requests.

REFERENCES

[1] Mohammed Amin Abdullah and Moez Draief. 2015. Global majority consensus by local majority polling on graphs of a given degree sequence. *Discrete Applied Mathematics* 180 (2015), 1–10.

[2] Vivek Bagaria, Sreeram Kannan, David Tse, Giulia Fanti, and Pramod Viswanath. [n. d.]. Deconstructing the Blockchain to Approach Physical Limits. https://arxiv.org/abs/1810.08092.

[3] S Basu, Ittay Eyal, and EG Sirer. [n. d.]. The Falcon Network.

[4] Alex Biryukov, Dmitry Khovratovich, and Ivan Pustogarov. 2014. Deanonymisation of clients in Bitcoin P2P network. In *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*. ACM, 15–29.

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

[6] Cardano. [n. d.]. Cardano Settlement Layer Documentation. https://cardanodocs.com/technical/.

[7] James Cruise and Ayalvadi Ganesh. 2014. Probabilistic consensus via polling and majority rules. *Queueing Systems* 78, 2 (2014), 99–120.

[8] Christian Decker and Roger Wattenhofer. 2013. Information propagation in the bitcoin network. In *Peer-to-Peer Computing (P2P), 2013 IEEE Thirteenth International Conference on*. IEEE, 1–10.

[9] Ittay Eyal, Adem Efe Gencer, Emin Gün Sirer, and Robbert Van Renesse. 2016. Bitcoin-NG: A Scalable Blockchain Protocol.. In *NSDI*. 45–59.

[10] Ittay Eyal and Emin Gün Sirer. 2018. Majority is not enough: Bitcoin mining is vulnerable. *Commun. ACM* 61, 7 (2018), 95–102.

[11] Giulia Fanti, Jiantao Jiao, Ashok Makkuva, Sewoong Oh, Ranvir Rana, and Pramod Viswanath. 2018. Barracuda: the Power of ℓ-polling in Proof-of-Stake Blockchains. (2018). available at http://swoh.web.engr.illinois.edu/polling.pdf and arXiv.

[12] MJ Fischer, N Lynch, and MS Paterson. 1985. Impossibility of Distributed Consensus with One Faulty Process.

[13] Juan Garay, Aggelos Kiayias, and Nikos Leonardos. 2015. The bitcoin backbone protocol: Analysis and applications. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 281–310.

[14] Yossi Gilad, Rotem Hemo, Silvio Micali, Georgios Vlachos, and Nickolai Zeldovich. 2017. Algorand: Scaling byzantine agreements for cryptocurrencies. In *Proceedings of the 26th Symposium on Operating Systems Principles*. ACM, 51–68.

[15] Johannes Göbel, Holger Paul Keeler, Anthony E Krzesinski, and Peter G Taylor. 2016. Bitcoin blockchain dynamics: The selfish-mine strategy in the presence of propagation delay. *Performance Evaluation* 104 (2016), 23–41.

[16] Ethan Heilman, Alison Kendler, Aviv Zohar, and Sharon Goldberg. 2015. Eclipse Attacks on Bitcoin's Peer-to-Peer Network.. In *USENIX Security Symposium*. 129–144.

[17] Aggelos Kiayias, Alexander Russell, Bernardo David, and Roman Oliynykov. 2017. Ouroboros: A provably secure proof-of-stake blockchain protocol. In *Annual International Cryptology Conference*. Springer, 357–388.

[18] Eleftherios Kokoris Kogias, Philipp Jovanovic, Nicolas Gailly, Ismail Khoffi, Linus Gasser, and Bryan Ford. 2016. Enhancing bitcoin security and performance with strong consistency via collective signing. In *25th USENIX Security Symposium (USENIX Security 16)*. 279–296.

[19] Timothy Lee. 2017. Bitcoin?s insane energy consumption, explained. (2017).

[20] Yoad Lewenberg, Yonatan Sompolinsky, and Aviv Zohar. 2015. Inclusive block chain protocols. In *International Conference on Financial Cryptography and Data Security*. Springer, 528–547.

[21] Chenxing Li, Peilun Li, Wei Xu, Fan Long, and Andrew Chi-chih Yao. 2018. Scaling Nakamoto Consensus to Thousands of Transactions per Second. *arXiv preprint arXiv:1805.03870* (2018).

[22] David Mazieres. 2015. The stellar consensus protocol: A federated model for internet-level consensus. *Stellar Development Foundation* (2015).

[23] Andrew Miller, James Litton, Andrew Pachulski, Neal Gupta, Dave Levin, Neil Spring, and Bobby Bhattacharjee. 2015. Discovering bitcoin?s public topology and influential nodes. *et al.* (2015).

[24] Michael Mitzenmacher and Eli Upfal. 2005. *Probability and computing: Randomized algorithms and probabilistic analysis*. Cambridge university press.

[25] Nikolaos Papadis, Sem Borst, Anwar Walid, Mohamed Grissa, and Leandros Tassiulas. 2018. Stochastic Models and Wide-Area Network Measurements for Blockchain Design and Analysis. In *IEEE INFOCOM 2018-IEEE Conference on Computer Communications*. IEEE, 2546–2554.

[26] Rafael Pass and Elaine Shi. 2017. Hybrid consensus: Efficient consensus in the permissionless model. In *LIPIcs-Leibniz International Proceedings in Informatics*, Vol. 91. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik.

[27] Rafael Pass and Elaine Shi. 2018. Thunderella: Blockchains with optimistic instant confirmation. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 3–33.

[28] Team Rocket. 2018. Snowflake to avalanche: A novel metastable consensus protocol family for cryptocurrencies,?

[29] Yonatan Sompolinsky, Yoad Lewenberg, and Aviv Zohar. 2016. SPECTRE: A Fast and Scalable Cryptocurrency Protocol. *IACR ePrint Archive* 2016 (2016), 1159.

[30] Yonatan Sompolinsky and Aviv Zohar. [n. d.]. PHANTOM. ([n. d.]).

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