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 at 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 canonical stochastic network model, we prove that this lightweight primitive strongly ameliorates the informational imbalance: the resulting throughput is as if the *entire* network were a factor of ℓ faster. We provide guidelines on how to implement Barracuda in practice, with a specific emphasis on proof-of-stake blockchains, guaranteeing robustness against several real-world factors.

KEYWORDS

Stochastic networks, blockchains

ACM Reference Format:
. 2019. Barracuda: The Power of ℓ-polling in Proof-of-Stake Blockchains . In *Proceedings of ACM Conference (Conference'17).* ACM, New York, NY, USA, 13 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

The word 'blockchain' has two meanings. On the one hand, a blockchain is 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 [7, 8, 12, 18, 19, 24], 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.

1.1 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*) [21], 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 a 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.

Conference'17, July 2017, Washington, DC, USA

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM.
https://doi.org/10.1145/nnnnnnn.nnnnnnn

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 [6], 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 [7] and Bloxroute [19]. The other is to design consensus algorithms that tolerate network latency by making use of forked branches. Examples include GHOST [32], SPECTRE [30], and Inclusive/Conflux [22]. In this paper, we design a P2P protocol called Barracuda that effectively reduces forking for a wide class of *existing* consensus algorithms.

1.2 Contributions

Our contributions in this work are threefold.

(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 [12]. Simulations under this model explain the gap observed in real-world cryptocurrencies, between the achievable block throughput 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 [16, 32]. We analyze the effect of average network delay on system throughput, and provide a lower bound on the block throughput.
(2) To mitigate the effect of forking from network delays, we propose a new block proposal algorithm called *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. Notably, this benefit emerges without actually changing any inherent network properties, like bandwidth or hardware. The analysis also has connections to load balancing in balls-and-bins problems, which is of independent interest.
(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 3; we analyze the block throughput of this model in Section 4. Next, we present Barracuda and analyze its block throughput in Section 5. Finally, we describe real-world implementation issues in Section 6, such as how to implement polling and analyzing adversarial robustness.

2 RELATED WORK

There are three main approaches in the literature for reducing forking. The first is to reduce the diversity of proposers. The second is to embrace forking, and use the forks to enhance throughput. The third is to algorithmically resolve forking before moving to the next block.

(1) Reducing proposer diversity. Forking is caused by the delay associated with the most recently-proposed block reaching the next proposer(s). A natural circumvention is to make the proposer(s) of consecutive blocks one and the same. This approach is proposed by Bitcoin-NG [13]: 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 worrying about network delay or forking. Although Bitcoin-NG has high throughput, it exhibits a few problems. First, whenever 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. Second, it suffers from high confirmation delay. To confirm a transaction in a longest-chain protocol, we require a threshold number of independently-selected proposers to append blocks to the chain containing that transaction. Since Bitcoin-NG elects a new proposer only once every epoch, this takes time comparable to Nakamoto consensus. Despite these problems, the idea of having a fixed proposer is also used in other protocols, such as Thunderella [27] and ByzCoin [20], which are also vulnerable to attacks on the proposer's IP address.

(2) Embracing forking. A different class of protocols has studied how to use forked blocktrees to contribute to throughput. Examples include GHOST [32], PHANTOM [31], SPECTRE [30], and Conflux [22]. GHOST describes a fork choice rule that tolerates honest forking by building on the heaviest subtree of the blocktree; it is described more carefully in Section 3.2. SPECTRE, PHANTOM, and Conflux instead use existing fork choice rules, but build a directed acyclic graph (DAG) over the produced blocks; this DAG is used to define an ordering over transactions. A formal understanding of these DAG-based protocols is still evolving; although the ideas are intuitively appealing, 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 [6]. 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. Our approach differs from [6] in that Barracuda is designed to be integrated into *existing* consensus protocols, whereas Prism is a completely new consensus protocol. Indeed, since each of the blocktrees in Prism uses the longest-chain fork choice rule, Barracuda can be used to reducing forking in each individual Prism blocktree.

(4) Fork-free consensus. Several consensus protocols tackle forking by preventing it entirely. Examples include Algorand [10, 17], Ripple [4], and Stellar [23]. These systems conduct a full round

of consensus for every block, e.g., using voting-based protocols. Disagreements about the next block are resolved immediately. Although voting-based consensus protocols consume additional time upfront for each block, the hope is that they improve overall efficiency by removing the need to resolve forks later; this hypothesis remains untested. A primary challenge in such protocols is that Byzantine-fault tolerant voting protocols can be communication-intensive, and require a known set of participants. Although there has been work addressing some of these challenges [20, 26], many industrial-grade blockchain systems running on BFT voting protocols require some degree of centralization for efficiency.

Our approach. Barracuda is complementary to these prior approaches in the sense that it provides a new networking protocol that consensus algorithms can benefit from. Our approach can be viewed as a partial execution of a polling-based consensus protocol. Polling has long been used in classical consensus protocols [5, 11, 15], as well as more recent work specific to blockchains [29]. Our approach differs from these algorithms in part because we do not use polling to reach complete consensus; rather we use it to reduce the number of inputs to a (separate) consensus protocol. Hence, Barracuda can be used as an add-on for many consensus protocols, especially those proposed for proof-of-stake (PoS) cryptocurrencies; we discuss these applications in Section 6.

3 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 delay Δ, proposer waiting time $\tilde{\Delta}$, and number of concurrent proposers k.

3.1 Modeling block generation

We model block generation as a discrete-time arrival process, where the t^{th} 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 [1], Qtum [3], and Particl [2]), 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 [9]). 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 3.1. 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 3.1.

In this work, we focus mainly on the PoS setting due to subtleties in the practical implementation of Barracuda (described in Section 5). 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 $\{(\text{Block } (j,i), \text{pointer to the parent block of } (j,i)) : 1 \le j \le t, 1 \le i \le k\}$ with the vertices being the union of the genesis block and all the blocks indexed as $\{(j,i) : 1 \le j \le t, 1 \le i \le k\}$.

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.

3.2 Network model and fork choice rule

We avoid modeling the topology of the underlying communication network by instead modeling the end-to-end delay of a message from any source to any destination node. In particular, we assume each transmission of a block over an edge experiences a 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 [12] 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 $\gamma(t)$, the proposer waits for time $\tilde{\Delta} \in [0, 1)$ to make a decision on where to append the new block in its local blocktree. The choice of parent block is governed by the local fork choice rule. Two of the most common fork choice rules are the Nakamoto protocol (longest chain) and the GHOST protocol.

- **Nakamoto protocol (longest chain).** When a node is elected as a proposer, the node attaches the block to the leaf of the *longest chain* in the *local* blocktree. When there is a tie (i.e., if there are multiple longest chains), the proposer chooses one arbitrarily. Longest chain is widely-used, including in Bitcoin, ZCash, and Monero.
- **GHOST protocol.** When a node is elected a proposer, the node attaches the block to the leaf of the *heaviest subtree* in the *local* blocktree. Concretely, the proposer starts from the genesis block and traverses the tree toward the leaves, until it reaches a leaf block. At each node, the proposer chooses the offspring with the largest number of descendants (i.e., the heaviest subtree). Ties are broken arbitrarily. A variant of the GHOST rule is used in Ethereum.

Both the Nakamoto and GHOST protocols belong 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 5 that our analysis applies generally to all local attachment protocols. In practice, almost all blockchains use local attachment protocols; however, some recent blockchains (e.g., Ripple) construct each new block through a cooperative process between a group of nodes.

Notice that if Δ is much smaller than the block inter-arrival time and all nodes obey protocol, then the global blocktree G_t will be a chain with high probability. On the other hand, if Δ is much larger than the block inter-arrival time, then G_t will be a star (i.e. a depth-one rooted tree) with high probability. 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.

4 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. More 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 [14] 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 [6, 16, 32].

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 [6, 16, 32], 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 1), we provide a lower bound on the block throughput, under the more nuanced network model from Section 3, and Nakamoto protocol fork-choice rule. This result holds for a deterministic arrival process. A proof is provided in Appendix 7.1.

Theorem 1. *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, and any average delay Δ, under*

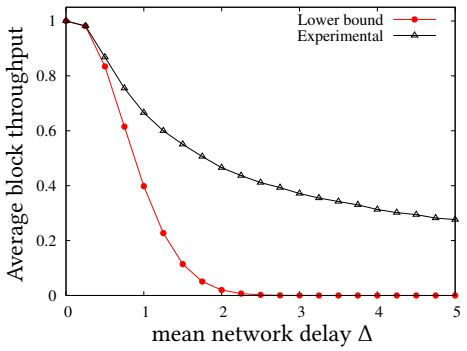

Figure 1: Block throughput vs. mean network delay for an inter-block time of 1 time unit.

the Nakamoto protocol, we have that

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

where $C_\Delta = e^{\frac{-1}{\Delta}}$.

Notice that trivially, $\mathbb{E}[L_{\mathsf{Chain}}(G_t)]/t \leq 1$, with equality when there is no network delay, $\Delta = 0$. Theorem 2 and our experiments in Figure 1 suggest that Theorem 1 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 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 $\exp\left(\frac{-C_\Delta}{(1-C_\Delta)^2}\right) = 1 - \frac{1}{t}$, 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 2. *Fix a confidence parameter* $\delta \in (0,1), k = 1, \gamma(t) = t$. *For both the Nakamoto and GHOST protocols, if*

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

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

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 that choose system parameters 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 Barracudathat effectively reduces forking without changing the system parameter Δ, which is determined by network bandwidth.

5 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 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. 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 6.1.

To investigate the effect of polling on the block chain, we define appropriate events on the probabilistic model of block arrival and block tree growth. We denote $X \sim \mathrm{Exp}(\lambda)$ 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 $B_{(j,i),m} \sim \mathrm{Exp}(1/\Delta)$. 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 $m^{(1)}_{(j,i)}$, and denote the other $\ell - 1$ nodes polled by node $m_{(j,i)}$ as $m^{(2)}_{(j,i)}, m^{(3)}_{(j,i)}, \ldots, m^{(\ell)}_{(j,i)}$.

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

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

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

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 $m^{(l)}_{(j,i)}$ 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

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

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 $\ell = 1$.

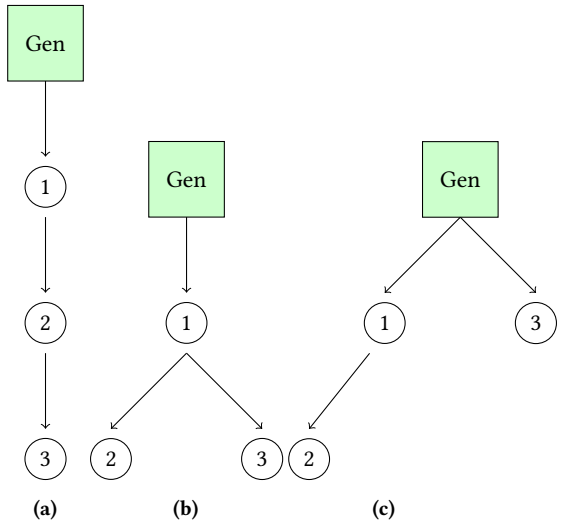

Figure 2: Examples of G_3 with varying structures.

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 2c, 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.

5.1 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, $\{m^{(l)}_{(j,i)} : j \in [t], i \in [k], l \in [\ell]\}$, 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 and GHOST protocols). We provide a proof in Section 7.3.

Theorem 3. *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]

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

We have the following results:

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

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

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

$$\mathsf{TV}\Big(P_{G_t}, P_{\tilde{G}_t}\Big) \leq \frac{(\ell k t)^2}{2n} . \tag{7}$$

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: $n \gg (\ell k t)^2$. 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 3.

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 3, the curves converge to a single curve, and are indistinguishable from one another. We used the network model from Section 3.2.

[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 $\{m^{(l)}_{(j,i)} : j \in [t], i \in [k], l \in [\ell]\}$ whenever all $tk\ell$ nodes in it are distinct.

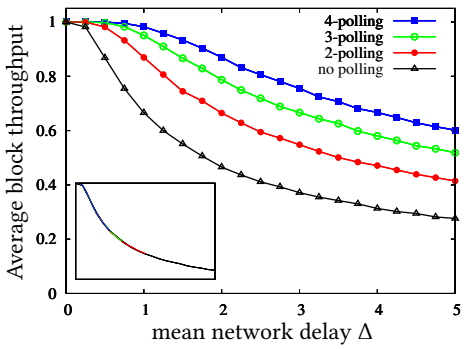

Figure 3: Comparing the average block throughput for various choices of ℓ confirms the theoretical prediction that ℓ-Barracuda effectively speeds up the network by a factor of ℓ; all curves are indistinguishable when x-axis is scaled as Δ/ℓ as shown in the inset.

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 $L_{\mathsf{Chain}}(G_t)$ to be the length of the longest chain in G_t excluding the Genesis block. Throughput is defined as $\mathbb{E}[L_{\mathsf{Chain}}(G_t)]/t$. We have the following Corollary of Theorem 3.

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

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

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.

5.2 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 $\Theta(\log t/\log\log t)$ [25]. 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 $\Theta(\log\log t/\log \ell)$ [25].

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 4. *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 provide a proof in Section 7.4. 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 stop, 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.

6 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 [12].

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 our experiments we use the Nakamoto longest chain fork choice rule. We ran our experiments for $T = 100$ time epochs on a network with $n = 10,000$ nodes with $k = 1$.

6.1 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 4 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 ('no setup delay' 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 5.

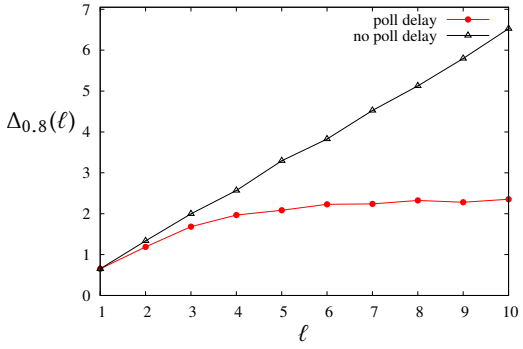

Figure 4: $\Delta_{0.8}(\ell)$ captures the highest delay Δ that achieves a desired block throughput of 0.8 under ℓ-Barracuda. With a polling delay of $\text{Exp}(1/(0.1\Delta))$, the performance saturates after $\ell = 6$ and eventually deteriorates at large ℓ.

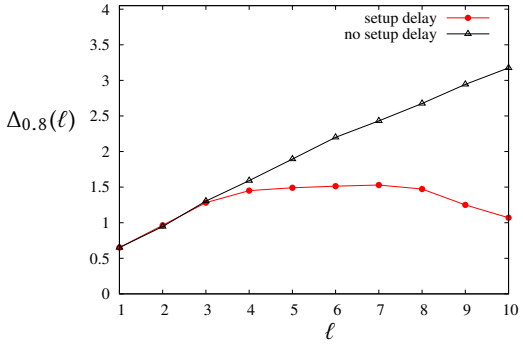

Figure 5: Average delay $\Delta_{0.8}(\ell)$ that achieves desired block throughput 0.8, when ℓ-Barracuda is used. We assume there is a polling delay of $\text{Exp}(1/(0.1\Delta))$ but the proposer waits exactly $\tilde{\Delta} = 0.1\Delta$ time before proposing. When there is no setup delay, polling more nodes is always better. Otherwise, we see an optimal ℓ, which depends on all system parameters.

6.2 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 the 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 their average network speed. The speed of a connection between two nodes is determined by the speed of the slower node. We compare the performance of ℓ-Barracuda with that of 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 6, we show the performance of a heterogeneous network with $h = 2$: half of the nodes have delay Δ and the other half have delay 5Δ. Every node has the same proposer election probability. ℓ-Barracuda gives a gain in the network throughput in line with the prediction of Theorem 3, even when the underlying network is heterogeneous.

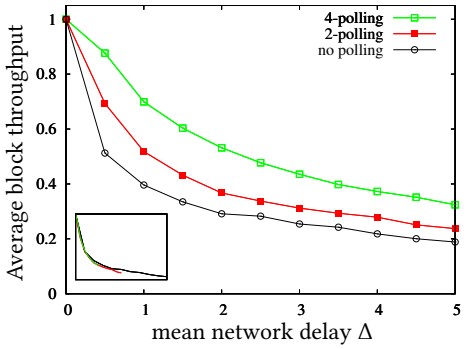

Figure 6: In a heterogeneous network, we enjoy the same polling gain as in a homogeneous network. Heterogeneous ℓ-Barracuda provides a speedup of the network by a factor of about ℓ, as shown in the inset where the x-axis is scaled by Δ/ℓ.

6.3 Polling partial blocktrees

The polling studied in this paper requires syncing of the complete local blocktree, 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 $t-b$. In Figure 7, we compare the performance of 2-polling and (2,1),(2,3),(2,5)-polling. The experiments suggest that comparable performance can be achieved with a choice of b that is small, and that choice increases with network delay.

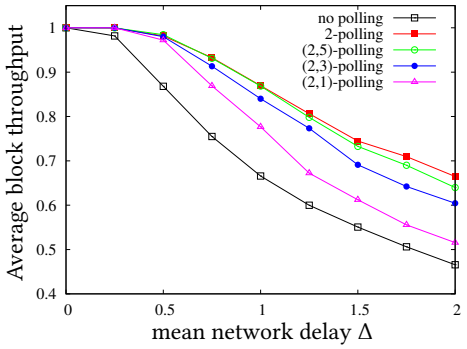

Figure 7: Sending the latest 5 blocks when polled is sufficient to achieve comparable performance as sending the entire local blocktree, while occupying significantly less network bandwidth.

6.4 Incentive Structure

To ensure timely response from polled nodes, we propose appropriate incentive mechanism, motivated by those used in BitTorrent. When BitTorrent nodes explore their neighbors to get new information, a reputation system is maintained, where nodes that upload have higher reputation, thus allowing for faster downloads. Incentive mechanisms for BitTorrent-like P2P networks have been analyzed to achieve Nash equilibrium as shown by [28]. We propose a reputation system for blockchain polling, where a node replies to a poll request only from a proposer with a higher reputation (tit-for-tat). A higher reputation is maintained by honestly responding to a polling request.

Another incentive mechanism involves the proposer paying polling fees to the the polled nodes. The fees can be in the form of a conditional payment, conditioned on the proposed block being finalized or reaching a depth of m blocks in the blockchain. Let P_c be the probability that block is finalized conditioned on the polled node sending its blocktree, and P_{nc} be the probability that block is finalized conditioned on the polled node sending nothing. Then the polled nodes should send its complete blocktree if:

$$R(P_c - P_{nc}) \quad > \quad C$$

where R is the polling fees/reward and C is the cost of sending blocktree to the proposer. Note that sending the blocktree only involves sending the difference of the blocktree of the polled node and the proposer, hence the cost C is relatively small. Sufficiently large fee R ensures that the nodes are incentivized to respond.

A systematic exploration of the parameters associated with the incentive mechanisms discussed above—along with a real-world system implementation of ℓ-Barracuda—is a topic of active research and outside the scope of this paper.

6.5 Security Implications

Blockchains are expected to operate in *permissionless* conditions, so 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. Since push and poll operations are different network primitives and since the poll operation is performed by proposer nodes, nodes when polled ascertain who the proposer of a new block. Such nodes could use this information perhaps initiate a denial of service attack on the proposer (or launch a bribery attack involving corrupting the proposer of the new block) – these possibilities are similar to (but more muted than) the attacks on the class of consensus protocols summarized under the item on "reducing proposer diversity" (example: Bitcoin-NG) and the vulnerabilities are no more than faced by this class of blockchain algorithms. However, a simple modification to the ℓ-Barracuda protocol nullifies even this relatively minor vulnerability.

Consider the following network protocol design symmetric with respect to polling: we replace push and poll network primitives by a single block tree information exchange primitive that we call "information-sync". Each connection/edge information-sync involves exchanging block tree information in a symmetric way between the pair of nodes. A new sync starts as soon as the previous

sync finishes. When a proposer is elected, it starts $\ell - 1$ new sync connections and proposes a block after completing sync from these $\ell - 1$ connections. As we can see those $\ell - 1$ nodes will consider this polling request as just another node asking for a sync, allowing the poll request to hide within the symmetrical structure. We tested this new protocol for the following parameters: $n = 500$ and $T = 100$ for a 4-regular graph with Exp(Δ) edge synchronization delays. The performance is shown in Figure 8, where we see that 5-polling offers a significant gains compared to no polling. Note that the overall gains are somewhat muted as compared to the polling gains on the pure ℓ-Barracuda protocol setting, as expected: the symmetric network protocol already includes some polling due to the nature of the information-sync primitive.

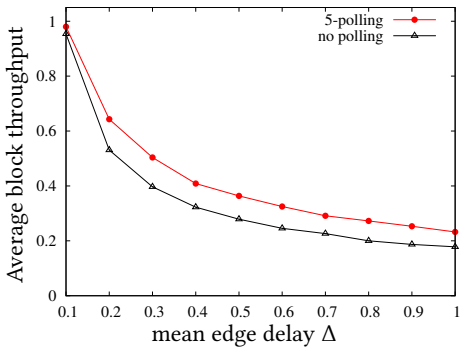

Figure 8: 5-polling shows gain over no polling in a homogeneous information sync network

Polling also helps when the network is heterogeneous, as shown in Figure 9, which uses the same node delay distribution as the heterogeneous exponential model discussed in section 4.2. A node's delay is defined as follows: the average edge synchronization delay between nodes with delay Δ_i and Δ_j is $\max(\Delta_i, \Delta_j)$ $\forall i \in [h]$. Here both the slower nodes and the faster nodes poll the same number of nodes.

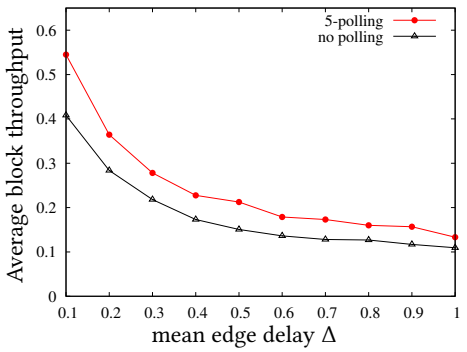

Figure 9: 5-polling shows gain over no polling in a heterogeneous information sync network

7 PROOFS OF THE MAIN RESULTS

We provide the proofs of the main results in this section.

7.1 Proof of Theorem 1

Since $k = 1$, we denote the proposer of block j as m_j. Thus the arrival time of the block j to node m is given by

$$R_{j,m} = \begin{cases} j & \text{if } m = m_j \\ j + B_{j,m} & \text{if } m \neq m_j \end{cases}.$$

Here $B_{j,m} \sim \text{Exp}(1/\Delta)$ and they are mutually independent random variables for $1 \leq j \leq t, 1 \leq m \leq n$. Define the event

$$e_{j,m,r} = \begin{cases} 1 & \text{when block } j \text{ is proposed, node } m \text{ has received block } r \\ 0 & \text{otherwise} \end{cases}$$

Thus whenever node m_t is chosen as a proposer for block t, the set of events $\{e_{t,m_t,r} : 1 \leq r \leq t-1\}$ determine the length of the longest chain $L_t \triangleq L_{\text{Chain}}(G_t)$. In particular, if we denote the blocks that are part of a longest chain $\text{Chain}(G_{t-1})$ as $j_1, \ldots, j_{L_{t-1}}$, we have that

$$L_t = L_{t-1} + 1, \quad \text{if } \prod_{r=1}^{L_{t-1}} e_{t,m_t,j_r} = 1.$$

In other words, had the node m_t received all the blocks that were part of a longest chain at time $t-1$, the length L_t would exceed L_{t-1} by 1. Since $L_t \geq L_{t-1}$ with probability 1, we obtain that

$$\mathbb{E}[L_t] \geq \mathbb{E}[L_{t-1}] + \mathbb{P}\left[e_{t,m_t,j_r} = 1, \forall r \in [L_{t-1}]\right].$$

Fix any $m_t \in [n]$. If m_t had not been a proposer for any of the blocks j_r, we get that

$$\mathbb{P}\left[e_{t,m_t,j_r} = 1, \forall r \in [L_{t-1}]\right] = \prod_{r=1}^{L_{t-1}} (1 - x^{t-j_r}), \quad x = e^{\frac{-1}{\Delta}}$$

Had m_t been a proposer before, clearly the above probability is still a lower bound on $\mathbb{P}\left[e_{t,m_t,j_r} = 1, \forall r \in [L_{t-1}]\right]$. Thus

$$\mathbb{E}[L_t] \geq \mathbb{E}[L_{t-1}] + \prod_{r=1}^{L_{t-1}} (1 - x^{t-j_r}) \geq \mathbb{E}[L_{t-1}] + \prod_{i=1}^{\infty} (1 - x^i).$$

Applying the above inequality recursively, we obtain that

$$\mathbb{E}[L_{\text{Chain}}(G_t)] \geq \left(\prod_{i=1}^{\infty} (1 - x^i)\right) \cdot t.$$

Thus is suffices to lower bound the Euler function $\phi(x) \triangleq \prod_{i=1}^{\infty}(1 - x^i)$. Using the inequality that $\frac{x}{1+x} \leq \log(1+x), x \geq -1$, we obtain that

$$\begin{aligned} \log \phi(x) &= \sum_{i \geq 1} \log(1 - x^i) = \sum_{i \geq 1} \frac{-x^i}{1 - x^i} \\ &\geq \sum_{i \geq 1} \frac{-x^i}{1 - x} = \frac{-x}{1 - x} \sum_{i \geq 0} x^i = \frac{-x}{(1-x)^2}. \end{aligned}$$

Thus $\phi(x) \geq \exp\left(\frac{-x}{(1-x)^2}\right)$.

7.2 Proof of Theorem 2

We apply Theorem 3 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 $\alpha \triangleq e^{\tilde{\Delta} l/\Delta}$, we have

$$\mathbb{E}[\mathbb{1}(E_{C,t,g})|\tilde{E}] = \prod_{j=2}^{t} \mathbb{E}[\mathbb{1}(e_{j,1} = 1, e_{j,2} = 1, \ldots, e_{j,j-1} = 1)|\tilde{E}] \quad (11)$$

$$= \prod_{j=2}^{t} \prod_{m=1}^{j-1} (1 - e^{\tilde{\Delta} l/\Delta} e^{-m\ell/\Delta}) \quad (12)$$

$$= \prod_{j=1}^{t-1} (1 - \alpha e^{-j\ell/\Delta})^{t-j} \quad (13)$$

$$\leq (1 - \alpha e^{-\ell/\Delta})^{t-1}. \quad (14)$$

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 $\alpha e^{-\ell/\Delta} \leq \frac{\ln\frac{1}{\delta}}{t}$. Hence,

$$\mathbb{E}[\mathbb{1}(E_{C,t,g})|\tilde{E}] \geq \prod_{j=1}^{t-1} \left(1 - \frac{c^j}{t^j}\right)^{t-j} \quad (15)$$

$$= \left(1 - \frac{c}{t}\right)^{t\frac{t-1}{t}} \prod_{j=2}^{t-1} \left(1 - \frac{c^j}{t^j}\right)^{t-j} \quad (16)$$

$$\overset{(a)}{\geq} e^{-c} - o(1) \quad (17)$$

$$= \delta - o(1), \quad (18)$$

where (a) follows from 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 $\alpha e^{-\ell/\Delta} \geq \frac{\ln\frac{1}{\delta}}{t}$, and

$$\mathbb{E}[\mathbb{1}(E_{C,t,g})|\tilde{E}] \leq (1 - c/t)^{t-1} \quad (19)$$

$$= e^{-c} + o(1) \quad (20)$$

$$= \delta + o(1). \quad (21)$$

Lemma 1. *Let $c > 0$ be fixed. Then we have that*

$$\lim_{n\to\infty} \sum_{k=2}^{n-1} (n-k) \log(1 - \frac{c^k}{n^k}) = 0.$$

Proof. Define $f_n(\cdot) : \mathbb{N} \to \mathbb{R}$ as

$$f_n(k) = (n-k)\log(1 - \frac{c^k}{n^k})\mathbb{1}\{2 \leq k \leq n-1\}.$$

For each fixed $k \in \mathbb{N}$, we have that $\lim_{n\to\infty} f_n(k) = 0$. Our goal is to show that $\lim_{n\to\infty} \int_{\mathbb{N}} f_n(k) d\mu(k) = 0$ where $\mu(\cdot)$ is the counting measure on $\mathbb{N}$. In view of dominated convergence theorem, hence it suffices to show that there exists a $g : \mathbb{N} \to \mathbb{R}$ such that

$$|f_n(k)| \leq g(k), \quad k \in \mathbb{N} \text{ and } \int g d\mu < \infty.$$

Note that for $2 \leq k \leq n-1$,

$$|f_n(k)| = (n-k)|\log(1 - \frac{c^k}{n^k})| \overset{(a)}{\leq} (n-k)\frac{\frac{c^k}{n^k}}{1 - \frac{c^k}{n^k}} \leq \frac{nc^k}{n^k - c^k},$$

where (a) follows from the fact that $|\log(1-x)| \leq \frac{x}{1-x}$ for $x \in [0,1]$. Let $n_0 \in \mathbb{N}$ be such that $n_0^k \geq 2c^k$. Hence, for $n \geq n_0$, we have

$$|f_n(k)| \leq \left(\frac{2}{c}\right)\left(\frac{c}{n}\right)^{k-1} \leq \left(\frac{2}{c}\right)\left(\frac{c}{n_0}\right)^{k-1} \triangleq g(k).$$

Clearly $\int_{\mathbb{N}} g(k) d\mu(k) < \infty$. The claim follows. □

7.3 Proof of Theorem 3

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

$$\mathbb{E}[e_{j,i,l,r}|\{m_{(j,i)}^{(l)} : j \in [t], i \in [k], l \in [\ell]\}, \{\gamma(i)\}_{i=1}^{t}] \quad (22)$$

$$= 1 - e^{-(\gamma(j) - \gamma(r[1]) - \tilde{\Delta})/\Delta}, \quad (23)$$

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

Denote the event of $\{m_{(j,i)}^{(l)} : j \in [t], i \in [k], l \in [\ell]\}$ are distinct as $\tilde{E}$. It follows from the definition of the local attachment protocol C that

$$\mathbb{E}[e_{j,i,l,r}|\tilde{E}, , \{\gamma(i)\}_{i=1}^{t}] = 1 - e^{-(\gamma(j) - \gamma(r[1]) - \tilde{\Delta})/\Delta} \quad (24)$$

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

$$e_{j,i,r} = 1 \Leftrightarrow \sum_{l \in [\ell]} e_{j,i,l,r} \geq 1, \quad (25)$$

it follows from the independence of $e_{j,i,l,r}$ and equation (22) that

$$\mathbb{E}[e_{j,i,r}|\tilde{E}, \{\gamma(i)\}_{i=1}^{t}] = 1 - e^{-(\gamma(j) - \gamma(r[1]) - \tilde{\Delta})\ell/\Delta} \quad (26)$$

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

Hence, we have

$$\mathbb{P}(\tilde{G}_t = g) = \mathbb{E}\left[\mathbb{1}(E_{C,t,g})|\tilde{E}, \{\gamma(i)\}_{i=1}^{t}\right] \quad (27)$$

$$= F(\{\gamma(i)\}_{i=1}^{t}, \frac{\Delta}{\ell}, \tilde{\Delta}, g, C). \quad (28)$$

Now we show the second part of Theorem 3. 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

$$\mathbb{P}(G_t \in A|\{\gamma(i)\}_{i=1}^t)$$
$$= \mathbb{E}\left[\sum_{i=1}^{A} \mathbb{1}(E_{C,t,g_i})\middle|\{\gamma(i)\}_{i=1}^t\right] \tag{29}$$
$$= \mathbb{P}(\tilde{E}|\{\gamma(i)\}_{i=1}^t)\mathbb{E}[\sum_{i=1}^{A} \mathbb{1}(E_{C,t,g_i})|\tilde{E},\{\gamma(i)\}_{i=1}^t]$$
$$+ (1-\mathbb{P}(\tilde{E}|\{\gamma(i)\}_{i=1}^t))\mathbb{E}[\sum_{i=1}^{A} \mathbb{1}(E_{C,t,g_i})|\tilde{E}^c,\{\gamma(i)\}_{i=1}^t] \tag{30}$$
$$= \mathbb{E}[\sum_{i=1}^{A} \mathbb{1}(E_{C,t,g_i})|\tilde{E},\{\gamma(i)\}_{i=1}^t]$$
$$+ (1-\mathbb{P}(\tilde{E}|\{\gamma(i)\}_{i=1}^t))$$
$$\times\left(\mathbb{E}[\sum_{i=1}^{A} \mathbb{1}(E_{C,t,g_i})|\tilde{E}^c,\{\gamma(i)\}_{i=1}^t] - \mathbb{E}[\sum_{i=1}^{A} \mathbb{1}(E_{C,t,g_i})|\tilde{E},\{\gamma(i)\}_{i=1}^t]\right) \tag{31}$$
$$= \mathbb{P}(\tilde{G}_t \in A)$$
$$+ (1-\mathbb{P}(\tilde{E}|\{\gamma(i)\}_{i=1}^t))$$
$$\times\left(\mathbb{E}[\sum_{i=1}^{A} \mathbb{1}(E_{C,t,g_i})|\tilde{E}^c,\{\gamma(i)\}_{i=1}^t] - \mathbb{E}[\sum_{i=1}^{A} \mathbb{1}(E_{C,t,g_i})|\tilde{E},\{\gamma(i)\}_{i=1}^t]\right). \tag{32}$$

It follows from the birthday paradox compution [25, Pg. 92] that

$$1-\mathbb{P}(\tilde{E}|\{\gamma(i)\}_{i=1}^t) \le \frac{kt\ell(kt\ell-1)}{2n}, \tag{33}$$

Hence, we have shown that for any measurable set A that G_t or $\tilde{G}_t$ take values in, we have

$$|\mathbb{P}(G_t \in A|\{\gamma(i)\}_{i=1}^t) - \mathbb{P}(\tilde{G}_t \in A)| \le 1-\mathbb{P}(\tilde{E}|\{\gamma(i)\}_{i=1}^t) \tag{34}$$
$$\le \frac{kt\ell(kt\ell-1)}{2n}. \tag{35}$$

The result follows from the definition of the total variation distance

$$\mathsf{TV}(P_{G_t|\{\gamma(i)\}_{i=1}^t}, P_{\tilde{G}_t}) = \sup_A |\mathbb{P}(G_t \in A|\{\gamma(i)\}_{i=1}^t) - \mathbb{P}(\tilde{G}_t \in A)|. \tag{36}$$

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:

$$\mathsf{TV}(P,Q) = \sup_{f:|f|\le\frac{1}{2}} \mathbb{E}_P f - E_Q f, \tag{37}$$

and taking $f = \frac{1}{t}\cdot(L_{\mathsf{Chain}}(G_t) - t/2)$, upon noticing that the length of the longest chain in the tree G_t is at most t.

7.4 Proof of Theorem 4

We index all the t balls as $1, 2, \ldots, t$. Denote the load of the maximally loaded bin after the placement of all balls at L_t. We aim at upper bounding

$$\mathbb{P}(L_t \ge k). \tag{38}$$

for $k \ge C\cdot\ell\cdot\frac{\log t}{\log\log t}$. It follows from the union bound that

$$\mathbb{P}(L_t \ge k) \le \sum_{j\in[t]} \sum_{k-\text{subset } (i_1,i_2,\ldots,i_k)\in[t]^k} \mathbb{P}(\text{Bin } j \text{ has balls } (i_1,i_2,\ldots,i_k)).$$

Hence, it suffices to upper bound each individual term. Note that in the placement of each ball, the probability that a specific bin was selected as a potential candidate is at most $\frac{\binom{t-1}{\ell-1}}{\binom{t}{\ell}} = \frac{\ell}{t}$. Hence, for each k-subset $(i_1, i_2, \ldots, i_k)$, the probability that bin j contains this k-subset is at most $\left(\frac{\ell}{t}\right)^k$. Applying the union bound, we have

$$\mathbb{P}(L_t \ge k) \le t\binom{t}{k}\left(\frac{\ell}{t}\right)^k.$$

Using the fact that $\left(\frac{n}{e}\right)^n \le n! \le e\sqrt{n}\left(\frac{n}{e}\right)^n$ for $n \ge 1$, we have

$$t\binom{t}{k}\left(\frac{\ell}{t}\right)^k \le t\frac{e\sqrt{t}t^t}{(t-k)^{t-k}k^k}\frac{\ell^k}{t^k}$$
$$\le et^{3/2}\left(\frac{t}{t-k}\right)^{t-k}\left(\frac{\ell}{k}\right)^k$$
$$\le et^{3/2}\left(1+\frac{k}{t-k}\right)^{\frac{t-k}{k}\cdot k}\left(\frac{\ell}{k}\right)^k$$
$$\le et^{3/2}\left(\frac{e\ell}{k}\right)^k.$$

Setting $et^{3/2}\left(\frac{e\ell}{k}\right)^k \le \frac{1}{t}$, the theorem is proved.

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 is a simple, lightweight method of improving throughput without needing to substantially alter either the underlying consensus protocol or the properties of 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 6.5), 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. Preliminary results suggest that this effect is small, but a formal analysis is needed to make this precise; this is an active area of research.

REFERENCES

[1] [n. d.]. Cardano. https://cardano.com/.
[2] [n. d.]. Particl. https://particl.com/.
[3] [n. d.]. Qtum. https://qtum.com/.
[4] [n. d.]. Ripple. https://ripple.com/.

[5] 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.

[6] 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.

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

[8] 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.

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

[10] J. Chen and S. Micali. 2016. Algorand. *arXiv:1607.01341* (2016).

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

[12] 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.

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

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

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

[16] 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.

[17] 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.

[18] 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.

[19] Uri Klarman, Soumya Basu, Aleksandar Kuzmanovic, and Emin Gün Sirer. [n. d.]. bloXroute: A Scalable Trustless Blockchain Distribution Network WHITEPAPER. ([n. d.]).

[20] 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.

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

[22] 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).

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

[24] 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).

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

[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] Dongyu Qiu and Rayadurgam Srikant. 2004. Modeling and performance analysis of BitTorrent-like peer-to-peer networks. In *ACM SIGCOMM computer communication review*, Vol. 34. ACM, 367–378.

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

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

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

[32] 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.