Prism: Deconstructing the Blockchain to Approach Physical Limits

Vivek Bagaria
vbagaria@stanford.edu
Stanford University

Sreeram Kannan
ksreeram@uw.edu
University of Washington at Seattle

David Tse
dntse@stanford.edu
Stanford University

Giulia Fanti
gfanti@andrew.cmu.edu
Carnegie Mellon University

Pramod Viswanath
pramodv@illinois.edu
University of Illinois at Urbana-Champaign

ABSTRACT

The concept of a blockchain was invented by Satoshi Nakamoto to maintain a distributed ledger. In addition to its security, important performance measures of a blockchain protocol are its transaction throughput and confirmation latency. In a decentralized setting, these measures are limited by two underlying physical network attributes: communication capacity and speed-of-light propagation delay. In this work we introduce Prism, a new proof-of-work blockchain protocol, which can achieve 1) security against up to 50% adversarial hashing power; 2) optimal throughput up to the capacity C of the network; 3) confirmation latency for honest transactions proportional to the propagation delay D, with confirmation error probability exponentially small in the bandwidth-delay product CD; 4) eventual total ordering of all transactions. Our approach to the design of this protocol is based on *deconstructing* Nakamoto's blockchain into its basic functionalities and systematically scaling up these functionalities to approach their physical limits.

ACM Reference Format:
Vivek Bagaria, Sreeram Kannan, David Tse, Giulia Fanti, and Pramod Viswanath. 2019. Prism: Deconstructing the Blockchain to Approach Physical Limits. In *2019 ACM SIGSAC Conference on Computer and Communications Security (CCS '19), November 11–15, 2019, London, United Kingdom.* ACM, New York, NY, USA, 18 pages. https://doi.org/10.1145/3319535.3363213

1 INTRODUCTION

In 2008, Satoshi Nakamoto invented the concept of a blockchain, a mechanism to maintain a distributed ledger in a permissionless setting. Honest nodes mine blocks on top of each other by solving Proof-of-Work (PoW) cryptographic puzzles; by following a longest chain protocol, they can come to consensus on a transaction ledger that is difficult for an adversary to alter. Since then, many other blockchain protocols have been invented.

CCS '19, November 11–15, 2019, London, United Kingdom

ACM ISBN 978-1-4503-6747-9/19/11…$15.00
https://doi.org/10.1145/3319535.3363213

1.1 Performance measures

The fundamental performance measures of a PoW blockchain protocol are:

(1) the fraction β of hashing power the adversary can control without compromising system security, assuming the rest of the nodes follow protocol;
(2) the throughput λ, number of transactions confirmed per second;
(3) the confirmation latency, τ, in seconds, for a given probability ε that a confirmed transaction will be removed from the ledger in the future.

For example, Bitcoin is secure against an adversary holding up to 50% of the total network hash power ($\beta = 0.5$), has throughput λ of a few transactions per seconds and confirmation latency of the order of tens of minutes to hours. There is a tradeoff between the confirmation latency and the confirmation error probability: the smaller the desired confirmation error probability, the longer the needed latency is in Bitcoin. For example, Nakamoto's calculations [16] show that for $\beta = 0.3$, while it takes a latency of 6 blocks (on the average, 60 minutes) to achieve an error probability of 0.15, it takes a latency of 30 blocks (on the average, 300 minutes) to achieve an error probability of 10^{-4}.

1.2 Physical limits

Bitcoin has strong security guarantees but its throughput and latency performance are poor. In the past decade, much effort has been expended to improve the performance in these metrics. But what are the fundamental bounds that limit the performance of *any* blockchain protocol?

Blockchains are protocols that run on a distributed set of nodes connected by a physical network. As such, their performance is limited by the attributes of the underlying network. The two most important attributes are C, the communication capacity of the network, and D, the speed-of-light propagation delay across the network. Propagation delay D is measured in seconds and the capacity C is measured in transactions per second. Nodes participating in a blockchain network need to communicate information with each other to reach consensus; the capacity C and the propagation delay D limit the *rate* and *speed* at which such information can be communicated. These parameters encapsulate the effects of both fundamental network properties (e.g., hardware, topology), as well as resources consumed by the network's relaying mechanism, such

as validity checking of transactions or blocks. [1] Assuming that each transaction needs to be communicated at least once across the network, it holds that λ, the number of transactions which can be confirmed per second, is at most C, i.e.

$$\lambda < C. \tag{1}$$

One obvious constraint on the confirmation latency τ is that

$$\tau > D. \tag{2}$$

Another less obvious constraint on the confirmation latency comes from the network capacity and the reliability requirement ε. Indeed, if the confirmation latency is τ and the block size is B_v transactions, then at most $C/B_v \cdot \tau$ mined blocks can be communicated across the network during the confirmation period for a given transaction. These mined blocks can be interpreted as confirmation *votes* for a particular transaction during this period; i.e. votes are communicated at rate C/B_v and $C\tau/B_v$ votes are accumulated over duration τ. (The parameter B_v can be interpreted as the minimum block size to convey a vote.) On average, a fraction $\beta < 0.5$ of these blocks are adversarial, but due to the randomness in the mining process, there is a probability, exponentially small in $C\tau/B_v$, that there are more adversarial blocks than honest blocks; if this happens, confirmation cannot be guaranteed. Hence, this probability is a lower bound on the achievable confirmation error probability, i.e. $\varepsilon = exp(-O(C\tau/B_v))$. Turning this equation around, we have the following lower bound on the latency for a given confirmation probability ε:

$$\tau = \Omega\left(\frac{B_v}{C} \cdot \log\frac{1}{\varepsilon}\right). \tag{3}$$

Comparing the two constraints, we see that if

$$\frac{CD}{B_v} \gg \log\frac{1}{\varepsilon},$$

the latency is limited by the propagation delay; otherwise, it is limited by the confirmation reliability requirement. The quantity CD/B_v is analogous to the key notion of *bandwidth-delay product* in networking (see eg. [11]); it is the number of "in-flight" votes in the network.

To evaluate existing blockchain systems with respect to these limits, consider a global network with communication links of capacity 20 Mbits/second and speed-of-light propagation delay D of 1 second. If we take a vote block of size 100 bytes, then the bandwidth-delay product $CD/B_v = 25000$ is very large. Hence, the confirmation latency is limited by the propagation delay of 1 seconds, but not by the confirmation reliability requirement unless it is astronomically small. Real-world blockchains operate far from these physical network limits. Bitcoin, for example, has λ of the order of 10 transactions per second, τ of the order of minutes to hours, and is limited by the confirmation reliability requirement rather than the propagation delay. Ethereum has $\lambda \approx 15$ transactions per second and $\tau \approx 3$ minutes to achieve an error probability of 0.04 for $\beta = 0.3$ [4].

[1] We define confirmation formally in Section 2, but informally, we say a node ε-confirms a transaction if, upon successfully evaluating a *confirmation rule* under parameter ε, the transaction has a probability of at most ε of being reverted by any adversary.

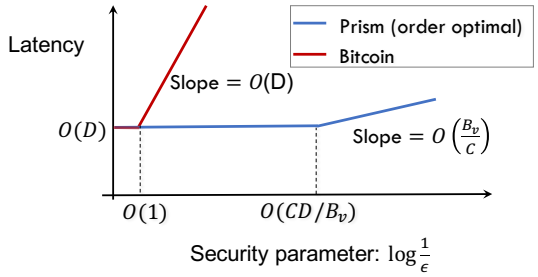

Figure 1: Confirmation latency vs. security parameter for Prism . The latency of Prism is independent of the security parameter value up to order CD/B_v and increases very slowly after that (with slope B_v/C). For Bitcoin , latency increases much more rapidly with the security parameter, with slope proportional to D. (Since $CD/B_v \gg 1$, this latter slope is much larger.)

1.3 Main contribution

The main contribution of this work is a new blockchain protocol, Prism, which, in the face of any powerful adversary[2] with power $\beta < 0.5$, can *simultaneously* achieve:

(1) **Security:** (Theorem 4.3) a total ordering of the transactions, with consistency and liveness guarantees.
(2) **Throughput:** (Theorem 4.4) a throughput

$$\lambda = 0.9(1-\beta)C \quad \text{transactions per second.} \tag{4}$$

(3) **Latency:** (Theorem 4.8) confirmation of all honest transactions (without public double spends) with an expected latency of

$$\mathbb{E}[\tau] < \max\left\{c_1(\beta)D, c_2(\beta)\frac{B_v}{C}\log\frac{1}{\varepsilon}\right\} \quad \text{seconds,} \tag{5}$$

with confirmation reliability at least $1-\varepsilon$ (Figure 1). Here, $c_1(\beta)$ and $c_2(\beta)$ are constants depending only on β

Notice that the worst-case optimal throughput of any protocol with $1-\beta$ fraction of hash power is $(1-\beta)C$ transactions/second, assuming each transaction needs to be communicated across the network. Hence, Prism's throughput is near-optimal. At the same time, Prism achieves a confirmation latency for honest transactions matching the two physical limits (2) and (3). In particular, if the desired security parameter $\log\frac{1}{\varepsilon} \ll CD/B_v$, the confirmation latency is of the order of the propagation delay and *independent* of $\log 1/\varepsilon$. Put another way, one can achieve latency close to the propagation delay with a confirmation error probability exponentially small in the bandwidth-delay product CD/B_v. Note that the latency is worst-case over all adversarial strategies but averaged over the randomness in the mining process.

To the best of our knowledge, no other existing PoW protocol has guaranteed performance which can match that of Prism. Two novel ideas which enable this performance are 1) a *total decoupling* of transaction proposing, validation and confirmation functionalities in the blockchain, allowing performance scaling; 2) the concept of confirming a *list* of possible ledgers rather than a unique ledger, enabling honest non-double-spend transactions to be confirmed quickly[3].

[2] The powerful adversary will be precisely defined in the formal model.
[3] This idea was inspired by the concept of list decoding from information theory.

1.4 Performance of existing PoW protocols

High forking protocols. Increasing the mining rate in Bitcoin can decrease latency and improve throughput, however, this comes at the expense of decreased security [25]. Thus, unlike Prism, the throughput and latency of Bitcoin is *security-limited* rather than *communication-limited*. To increase the mining rate while maintaining security, one line of work (GHOST [25], Inclusive [14], Spectre [23], Phantom [24], Conflux [15]) in the literature has used more complex fork choice rules and added reference links to convert the blocktree into a directed acyclic graph (DAG). This allows blocks to be voted on by blocks that are not necessarily its descendants.

While GHOST remains secure at low mining rates[10], there is a balancing attack by the adversary [12, 17], which severely limits the security at high mining rates. Thus, like Bitcoin, the throughput of GHOST is security-limited. The other protocols Inclusive and Conflux that rely on GHOST inherit this drawback. While Spectre and Phantom improve latency and throughput, Spectre cannot provide a total order on all transactions (required for smart contracts) and Phantom does not yet have a formal proof of security.

Decoupled consensus. Protocols such as BitcoinNG [7] decouple transaction proposal and leader election (which are coupled together in Bitcoin). BitcoinNG elects a single leader to propose many transaction blocks till the next leader is elected by PoW. While this achieves high throughput, the latency cannot be reduced using this approach. Furthermore, BitcoinNG is vulnerable to bribery or DDoS attacks, whereby an adversary can corrupt a leader after learning its identity (unlike Bitcoin). Subchains [22] and weak blocks [2, 26] both employ blocks with lower hash threshold ("weak blocks") along with regular blocks in an attempt to scale throughput. However, since weak blocks are required to form a chain, it does not achieve the optimal throughput.

Hybrid blockchain-BFT consensus. Several protocols combine ideas from Byzantine fault tolerant (BFT) based consensus into a PoW setting [1, 13, 20, 21]. ByzCoin [13] and its predecessor PeerCoin [6] attempt to address the latency shortcoming of BitcoinNG but is proven in a later paper [20] to be insecure when the adversarial fraction $\beta > 0.25$. *Hybrid consensus* uses a combination of proof-of-work based committee selection with Byzantine fault tolerance (BFT) consensus [20]. However, this protocol is secure only till $\beta = 0.33$. While the protocol latency is responsive, i.e., it decreases with network delay linearly, for a known network delay, it has similar non-optimal dependence on ε as Bitcoin.

A closely-related protocol called Thunderella [21] achieves very low latency under optimistic conditions, i.e., when the leader is honest and $\beta < 0.25$. However even when β is very small, a dishonest leader can keep delaying transactions to the Bitcoin latency (since such delaying behavior is detected by a slow PoW blockchain).

1.5 Our Approach

Increasing the mining rate is critical to improving the throughput and latency of blockchain protocols. The challenges facing the DAG approaches arise from the fact that the DAG is *unstructured*, due to the excessive random forking when the mining rate is increased. In contrast, Prism is based on a *structured* DAG created by cryptographic sortition of the mined blocks into different types of different functionalities and scaling these functionalities separately.

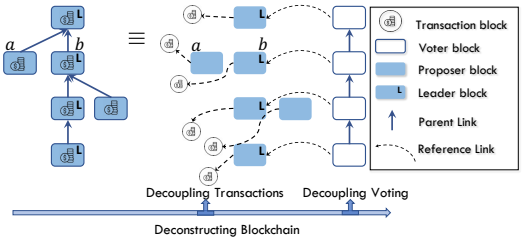

Figure 2: Deconstructing the blockchain into transaction blocks, partially ordered proposal blocks arranged by level, and voter blocks organized in a voter tree. The main chain is selected through voter blocks, which vote among the proposal blocks at each level to select a leader block. For example, at level 3, block b is elected the leader over block a.

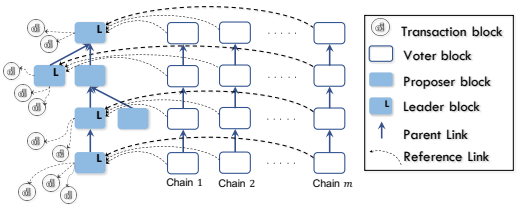

Figure 3: Prism. Throughput, latency and reliability are scaled to the physical limits by increasing the number of transaction blocks and the number of parallel voting chains.

Deconstruction. We start by deconstructing the basic blockchain structure into its atomic functionalities, illustrated in Figure 2. The selection of a main chain in a blockchain protocol (e.g., the longest chain in Bitcoin) can be viewed as electing a leader block among all the blocks at each level of the blocktree, where the level of a block is defined as its distance (in number of blocks) from the genesis block. Blocks in a blockchain then serve three purposes: they stand for election to be leaders, they add transactions to the main chain, and they vote for ancestor blocks through parent link relationships. We explicitly separate these three functionalities by representing the blocktree in a conceptually equivalent form (Figure 3). In this representation, blocks are divided into three types: proposer blocks, transaction blocks and voter blocks. The voter blocks vote for transactions indirectly by voting for proposer blocks, which in turn link to transaction blocks . Proposer blocks are grouped according to their level in the original blocktree, and each voter block votes among the proposer blocks at the same level to select a leader block among them. The elected leader blocks can then bring in the transactions to form the final ledger. The valid voter blocks are the ones in the longest chain of the voter tree, and this longest chain maintains the security of the whole system.

Scaling. This alternative representation of the traditional blockchain, although seemingly more complex than the original blockchain representation, provides a natural path for scaling performance to approach physical limits (Figure 3). To increase the transaction throughput, one can simply increase the number of transaction blocks that a proposer block points to without compromising the security of the blockchain. This number is limited only by the physical capacity of the underlying communication network. To provide fast confirmation, one can increase the number of parallel voting trees, voting on the proposal blocks in parallel to increase the voting

rate, until reaching the physical limit of confirming with speed-of-light latency and extremely high reliability. Note that even though the overall block generation rate has increased tremendously, the number of proposal blocks per level remains small and manageable, and the voting blocks are organized into many separate voting chains with low block mining rate per chain and hence little forking. The overall structure, comprising of the different types of blocks and the links between them, is a structured DAG.

Sortition. The sortition of blocks into the three types of blocks, and further into blocks of different voting trees, can be accomplished by using the random hash value when a block is successfully mined. This sortition splits the adversary power equally across the structures and does not allow it to focus its power to attack specific structures. This sortition is similar to the 2-for-1 PoW technique used in [9], which is also used in Fruitchains [19] for the purpose of providing fairness in rewards. In fact, the principle of *decoupling* functionalities of the blockchain, central to our approach, has already been applied in Fruitchains, as well as other works such as BitcoinNG. The focus of these works is only on decoupling the transactions-carrying functionality. In our work, we broaden this principle to decouple *all* functionalities.

Concurrent work. We were made aware of two independent but related works [8, 27] which appeared after we posted this work online. [8] proposes two protocols, one achieves high throughput $O(C)$ but Bitcoin latency, and the other achieves low latency $O(1/\sqrt{C})$ but low throughput $O(1)$. In contrast, Prism achieves *simultaneously* high throughput $O(C)$ and even lower latency $O(1/C)$. Although [8] also uses the concept of multiple chains, the key difference with Prism is that there is *no* decoupling: the blocks in each chain both carry transactions and vote. Thus, either different transactions are put on the different chains to increase throughput, but the voting rate is low and hence the latency is poor, or the same transaction is repeated across all the chains to increase the voting rate, but the throughput is poor. In contrast, Prism decouples blocks into transaction blocks and voter blocks, tied together through proposer blocks, and allocate a fraction of the network capacity to each to deliver both low latency and high throughput. The protocol in [27] is similar to first one in [8], achieving high throughput but only Bitcoin latency.

1.6 Outline of paper

Section 2 presents our model. It is a combination of the synchronous model used in [9] and a network model that ties the blockchain parameters to physical parameters of the underlying network. In Section 3, we give a pseudocode description of Prism. The analysis of the security, throughput and latency of Prism is outlined in Section 4 and the outline of the proofs are presented in Appendices D, E and F. The full proofs can be found in our extended manuscript due to space constraints [3]. Section 5 contains simulation results.

2 MODEL

We consider a synchronous, round-based network model similar to that of Garay *et al.* [9]. We define a blockchain protocol as a pair (Π, g), where Π is an algorithm that maintains a blockchain data structure C consisting of a set of *blocks*. The function $g(\mathsf{tx}, C)$ encodes a *ledger inclusion rule*; it takes in a transaction tx and a blockchain C, and outputs $g(\mathsf{tx}, C) = 1$ if tx is contained in the ledger defined by blockchain C and 0 otherwise. For example, in Bitcoin, $g(\mathsf{tx}, C) = 1$ iff tx appears in any block on the longest chain. If there are multiple longest chains, g can resolve ties deterministically, e.g., by taking the chain with the smallest hash value.

The blockchain protocol proceeds in rounds of Δ seconds each. Letting κ denote a security parameter, the *environment* $\mathcal{Z}(1^\kappa)$ captures all aspects external to the protocol itself, such as inputs to the protocol (i.e., new transactions) or interaction with outputs.

Let $\mathcal{N}$ denote the set of participating nodes. The set of *honest* nodes $\mathcal{H} \subset \mathcal{N}$ strictly follow the blockchain protocol (Π, f). *Corrupt* nodes $\mathcal{N} \setminus \mathcal{H}$ are collectively controlled by an adversarial party $\mathcal{A}$. Both honest and corrupt nodes interact with a random function $H : \{0,1\}^* \to \{0,1\}^\kappa$ through an oracle $\mathsf{H}(x)$, which outputs $H(x)$. In each round, each node $n \in \mathcal{N}$ is allowed to query the oracle $\mathsf{H}(\cdot)$ at most q times. The adversary's corrupt nodes are collectively allowed up to $\beta q|\mathcal{N}|$ sequential queries to oracle $\mathsf{H}(\cdot)$, where $\beta < 0.5$ denotes the fraction of adversarial hash power, i.e., $1 - \frac{|\mathcal{H}|}{|\mathcal{N}|} = \beta$.[4] Like [9], the environment is not allowed to access the oracle. These restrictions model the limited hash rate in the system.

In an execution of the blockchain protocol, the environment $\mathcal{Z}$ first initializes all nodes as either honest or corrupt; like [9], once the nodes are initialized, the environment can adaptively change the set $\mathcal{H}$ between rounds, as long as the adversary's total hash power remains bounded by β. Thereafter, the protocol proceeds in rounds. In each round, the environment first delivers inputs to the appropriate nodes (e.g., new transactions), and the adversary delivers any messages to be delivered in the current round. Here, delivery means that the message appears on the recipient node's input tape. Nodes incorporate the inputs and any messages (e.g., new blocks) into their local blockchain data structure according to protocol Π. The nodes then access the random oracle $\mathsf{H}(\cdot)$ as many times as their hash power allocation allows. Hence, in each round, users call the oracle $\mathsf{H}(\cdot)$ with different nonces s in an attempt to find a valid proof of work. If an oracle call produces a proof of work, then the node can deliver a new block to the environment. Note that the computational constraints on calling oracle $\mathsf{H}(\cdot)$ include block validation. Since each block only needs to be validated once, validation represents a small fraction of computational demands.

Since each node is allowed a finite number of calls to $\mathsf{H}(\mathsf{x})$ in each round, the number of blocks mined per round is a Binomial random variable. To simplify the analysis, we consider a limit of our model as the number of nodes $|\mathcal{N}| \to \infty$. As $|\mathcal{N}|$ grows, the proof-of-work threshold adjusts such that the expected number of blocks mined per round remains constant. Hence, by the Poisson limit theorem, the number of voter blocks mined per round converges to a Poisson random variable.

All messages broadcast to the environment are delivered by the adversary. The adversary has various capabilities and restrictions. (1) Any message broadcast by an honest node in the previous round must be delivered by the adversary at the beginning of the current round to all remaining honest nodes. However, during delivery, the adversary can present these messages to each honest node in whatever order it chooses. (2) The adversary cannot forge or alter

[4] β for **b**ad. Like [9], we have assumed all nodes have the same hash power, but this model can easily be generalized to arbitrary hash power distributions.

any message sent by an honest node. (3) The adversary can control the actions of corrupt nodes. For example, the adversary can choose how corrupt nodes allocate their hash power, decide block content, and release mined blocks. Notably, although honest blocks publish mined blocks immediately, the adversary may choose to keep blocks they mined private and release in future round. (4) The adversary can deliver corrupt nodes' messages to some honest nodes in one round, and the remaining honest nodes in the next round. We consider a "rushing" adversary that observes the honest nodes' actions before taking its own action for a given round. Notice that we do not model rational users who are not necessarily adversarial but nevertheless may have incentives to deviate from protocol.

Physical Network Constraints. To connect to the physical parameters of the network, we assume a simple network model. Let B be the size of a block, in units of number of transactions. The network delay Δ (in seconds) is given by:

$$\Delta = \frac{B}{C} + D \tag{6}$$

i.e. there is a processing delay of B/C followed by a propagation delay of D seconds. This is the same model used in [25], based on empirical data in [5], as well in [22]. Notice that the network delay Δ is by definition equal to the duration of a single round.

In practice, networks cannot transport an infinite number of messages at once. We model this by allowing the environment to transport only a finite volume of messages per round. This volume is parametrized by the *network capacity* C, measured in units of transactions per second. Hence, during each round, the environment can process a message volume equivalent to at most ΔC transactions. This puts a constraint on the number of blocks mined per unit time in any protocol. This *stability constraint* differentiates our model from prior work, which has traditionally assumed infinite network capacity; in particular, this gives us a foothold for quantifying physical limits on throughput and latency.

For simplicity, we assume that the dissemination of new transactions consumes no bandwidth. Instead, the cost of communicating transaction messages is captured when the environment transmits blocks carrying transactions. In other words, we assume that the cost of transmitting transactions is counted only once.

Metrics. We let random variable $\mathsf{VIEW}_{\Pi,\mathcal{A},\mathcal{Z}}$ denote the joint view of all parties over all rounds; here we have suppressed the dependency on security parameter κ. The randomness is defined over the choice of function $H(\cdot)$, as well as any randomness in the adversary $\mathcal{A}$ or environment $\mathcal{Z}$. Our goal is to reason about the joint view for all possible adversaries $\mathcal{A}$ and environments $\mathcal{Z}$. In particular, we want to study the evolution of C_i^r, or the blockchain of each honest node $i \in \mathcal{H}$ during round r. Following the Bitcoin backbone protocol model [9], we consider protocols that execute for a finite execution horizon $r_{\max}$, polynomial in κ. Our primary concern will be the efficiency of *confirming* transactions.

Definition 2.1. *We say a transaction* tx *is* $(\varepsilon, \mathcal{A}, \mathcal{Z}, r_0, \kappa)$*-cleared iff under an adversary* $\mathcal{A}$*, environment* $\mathcal{Z}$*, and security parameter* κ*,*

$$\mathbb{P}_{\mathsf{VIEW}_{\Pi,\mathcal{A},\mathcal{Z}}}\left(\bigcap_{\substack{r\in\{r_0,\ldots,r_{\max}\}\\ i\in\mathcal{H}}} \{g(\mathsf{tx}, C_i^r) = b\}\right) \geq 1 - \varepsilon - \mathsf{negl}(\kappa),$$

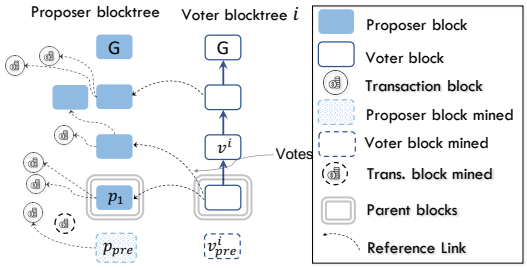

Figure 4: Snapshot of a miner's blocktree: The previously mined blocks have solid boundary whereas blocks which are being mined have dotted-boundary. A miner simultaneously mines on p_1, parent on proposer blocktree, v^i, parent on voter block blocktree $i(\forall i \in [m])$.

where $b \in \{0, 1\}$*;* $b = 1$ *corresponds to confirming the transactions and* $b = 0$ *corresponds to rejecting the transaction.*

That is, a transaction is considered confirmed (resp. rejected) if all honest party will include (resp. exclude) it from the ledger with probability more than ε plus a term negligible in κ resulting from hash collisions, which we ignore in our analysis. We suppress the notation κ from here on.

Our objective is to optimize two properties of a blockchain protocol: the throughput and latency of confirming transactions. We let $|S|$ denote the number of elements in set S. We let $\mathcal{T}$ denote the set of all transactions generated during the execution horizon, and $\mathcal{T}^r$ denote all transactions delivered up to and including round r.

Definition 2.2 (Throughput). *We say a blockchain protocol* Π *supports a throughput of* λ *transactions per round if there exists* U_ε*,linear in* $\log(1/\varepsilon)$*, such that for all environments* $\mathcal{Z}$ *that produce at most* λ *transactions per round, and for* $\forall\ r \in [1, r_{\max}]$*,*

$$\max_{\mathcal{A}} \left|\left\{\mathsf{tx} \in \mathcal{T}^r \ :\ \mathsf{tx} \text{ is not } (\varepsilon, \mathcal{A}, \mathcal{Z}, r)\text{-cleared}\right\}\right| < \lambda U_\varepsilon. \tag{7}$$

The system throughput *is the largest throughput that a blockchain protocol can support.*

Notice that although $|\mathcal{T}^r|$ grows with r, the right-hand side of (7) is constant in r; this implies that the system throughput λ is the expected *rate* at which we can clear transactions maintaining a bounded transaction queue, taken worst-case over adversary $\mathcal{A}$ and environments $\mathcal{Z}$ producing at most $\lambda\Delta$ transactions per round.

Definition 2.3 (Latency). *For a transaction* tx*, let* $r(\mathsf{tx})$ *denote the round in which the transaction was first introduced by the envioronment, and let random variable* $R_\varepsilon(\mathsf{tx})$ *denote the smallest round* r *for which* tx *is* $(\varepsilon, \mathcal{A}, \mathcal{Z}, r)$*-cleared. The expected* ε-latency *of transaction* tx *is defined as:*

$$\tau_\varepsilon(tx) \triangleq \max_{\mathcal{Z},\mathcal{A}} \ E_{\mathsf{VIEW}_{\Pi,\mathcal{A},\mathcal{Z}}}\left[R_\varepsilon(\mathsf{tx}) - r(\mathsf{tx})\right] \tag{8}$$

Note that if all transactions have finite ε-latency, it implies that the blockchain has both consistency and liveness properties.

3 PROTOCOL DESCRIPTION

We first describe the content and roles of three types of blocks in the Prism(Π, g) blockchain. We then present Algorithm 1, which defines the protocol Π and the blockchain data structure C. We

then define the *ledger inclusion rule*, g, in Algorithm 2. Due to space constraints, all pseudocode for these algorithms can be found in Appendix A. Prism's blockchain data structure, C, has one proposer blocktree and m voter blocktrees, as shown in Figure 3. We use these different blocktrees to maintain three distinct types of blocks:

Proposer blocks: Proposer blocks represent the skeleton of the Prism blockchain and are mined on the proposer blocktree according to the longest-chain rule. The *level* of a proposer block is defined as its distance from the proposer genesis block. The blocktree structure is only utilized in our protocol as a proof of level of a given proposal block. To construct the ledger, our protocol selects proposal block sequences where one block is chosen at each level. Proposer blocks can *refer* to transaction blocks and other proposer blocks by including pointers to referred blocks in their payload. For example, in Fig 4, the proposer blocktree has two proposer blocks mined at level 1, and one proposer block mined at levels 2 and 3, and they point to five transaction blocks in total.

Voter blocks: Voter blocks are mined on m separate voter blocktrees, each with its own genesis block, according to the longest chain rule. We say a voter block *votes* on a proposer block B if it includes a pointer to B in its payload. Note that unlike many BFT consensus protocols, a malicious miner in Prism cannot equivocate when voting because voter blocks are sealed by proof of work. Even if a miner mines conflicting voter blocks and tries to send them to disjoint sets of honest users, all users will receive both blocks within one round. Each longest chain from each voter blocktree can cast at most one vote for each level in the proposer blocktree. More precisely, a voter block votes on all levels in the proposer tree that are unvoted by the voter block's ancestors. Therefore, the voter trees collectively cast at most m votes on a given level of the proposer blocktree. Fig. 3 shows voter blocktree i and its votes (dotted arrows) on each level of the proposer blocktree. For each level ℓ on the proposer blocktree, the block with the highest number of votes is defined as the *leader* block of level ℓ.

Transaction blocks: Transaction blocks contain transactions and are mined on the proposer blocktree as in Fig. 3. Although transaction blocks are not considered part of the proposer blocktree, each transaction block has a proposer block as its parent.

The process by which a transaction is included in the ledger is as follows: (1) the transaction is included in a transaction block B_T. (2) B_T is referred by a proposer block B_P. (3) Proposer block B_P is confirmed, either directly (by becoming a leader) or indirectly (e.g., by being referred by a leader).

3.1 Protocol Π

Algorithm 1 presents Prism's protocol Π. The protocol begins with a trusted setup, in which the environment generates genesis blocks for the proposer blocktree and each of the m voter blocktrees. Once the trusted setup completes, the protocol enters the mining loop.

Whereas Bitcoin miners mine on a single blocktree, Prism miners simultaneously mine one proposer block, one transaction block, and m voter blocks, each with its own parent and content. This simultaneous mining happens via cryptographic sortition. Roughly, a miner first generates a "superblock" that contains enough information for all $m + 2$ blocks simultaneously. It then tries different nonce values; upon mining a block, the output of the hash is deterministically mapped to either a voter block (in one of the m trees), a transaction block, or a proposer block (lines 41-47 in Algorithm 1). After sortition, the miner discards unnecessary information and releases the block to the environment.

More precisely, while mining, each miner maintains outstanding content for each of the $m + 2$ possible mined blocks. In Bitcoin, this content would be the transaction memory pool, but since Prism has multiple types of blocks, each miner stores different content for each block type. For transaction blocks, the content consists of all transactions that have not yet been included in a transaction block. For proposer blocks, the content is a list of transaction blocks and proposer blocks that have not been referred by any other proposer block. For voter blocks in the ith voter tree, the content is a list of proposer blocks at each level in the proposer blocktree that has not yet received a vote in the longest chain of the ith voter tree. If a miner observes multiple proposer blocks at the same level, it always votes on the first one it received. For example, in Figure 4, voter block v^i_{new} votes on one proposer block on levels 3 and 4 because its ancestors have voted on level 1 and 2.

Upon collecting this content, the miner generates a block. Instead of naively including all the $m+2$ parents[5] and content hashes in the block, Prism's header contains a) the Merkle root of a tree with $m+2$ parent blocks, b) the Merkle root of a tree with $m + 2$ contents, and c) a nonce. Once a valid nonce is found, the block is sortitioned into a proposer block, a transaction block, or a voter block on one of the m voter trees. The mined, sortitioned block consists of the header, the appropriate parent and content, and their respective Merkle proofs. For instance, if the mined block is a proposer block, it would contain only the proposer parent reference, proposer content, and appropriate Merkle proofs; it would *not* store transactions or votes.

While mining, nodes may receive blocks from the network, which are processed in much the same way as Bitcoin. Upon receiving a new block, the miner first checks validity. A block B is valid if it satisfies the PoW inequality and the miner has all the blocks (directly or indirectly) referred by B. If the miner lacks some referred blocks, it requests them from the network. Upon receiving a valid transaction block B, the miner removes the transactions in B from its transaction pool and adds B to the unreferred transaction block pool. Upon receiving a valid voter block, the miner updates the longest chain if needed, and updates the vote counts accordingly. Upon receiving a valid proposer block B at a level ℓ higher than the previous highest level, the miner makes B the new parent proposer block, and updates all m voter trees to vote on B at level ℓ.

3.2 Ledger confirmation rule g

As defined before, the proposer block with the most votes on level ℓ is defined as the *leader block* of level ℓ. The leader block for a fixed level ℓ can initially fluctuate when the voter blocktrees start voting on level ℓ. However, as the voter blocktrees grow, these votes on level ℓ are cemented deeper into their respective voter blocktrees and the leader fluctuation ceases and thus we can confirm the leader block at level w.h.p. The sequence of leader blocks for each level of the proposer blocktree is defined as the *leader sequence*.

[5] Proposer and tx block share the same parent and are included twice for simplicity.

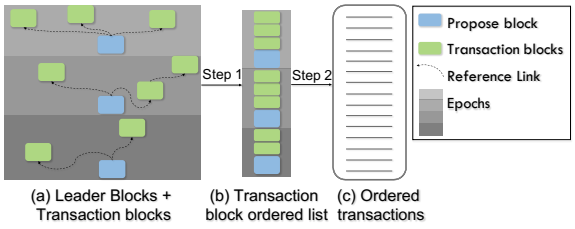

Figure 5: BuildLedger(): The proposer blocks for a given proposer block sequence are blue, and the referenced transaction blocks are green. Each shade of gray region is all the tx blocks referred by the proposer block.

Confirmation and Ordering: A set of transactions can often be individually confirmed before being ordered among themselves. For this reason, confirming transactions is easier than ordering the transactions. For example, consider the following two transactions a) Alice pays Bob \$10, and b) Carol pays Drake \$10. Both these transactions can be individually confirmed without deciding which transaction occurred first. In Bitcoin, transactions are simultaneously confirmed and ordered; however, in Prism, transactions can be confirmed before being ordered. The procedure IsTxConfirmed() in Algorithm 2 defines the transaction confirmation rule g and the procedure GetOrderedConfirmedTxs() defines the rule for ordering the confirmed transactions. Both these procedures use BuildLedger() which is described next.

BuildLedger(): Given a proposer block sequence from levels 1 to ℓ, $\{p_1, \cdots, p_\ell\}$, represented by blue blocks in Fig. 5(a). Let L_{p_i}, represented by green blocks in the grey shaded area in Fig. 5(a), be an ordered list of all the transaction blocks directly or indirectly referred by block p_i. Note that a transaction block t is indirectly referred by proposer block p_i if p_i includes a reference link to another proposer block p' that directly refers t. Since honest proposer blocks link to any unreferenced transaction blocks and proposer blocks, this ensures that the transaction blocks not referred by the proposer leader sequence are also included in the ledger. Let $\{L_{p_1}, \cdots, L_{p_\ell}\}$ be the *transaction block list* of sequence $\{p_1, \cdots, p_\ell\}$ as shown in Fig. 5(b). The procedure then expands this transaction-block list and remove all the duplicate and double-spent transactions to output *ordered-transaction list* as shown in Fig. 5(c).

IsTxConfirmed(): While confirming a leader block can take some time[6], we quickly narrow down a set of proposer blocks, defined as *proposer-set*, which is guaranteed to contain the leader block for that level. The proposer-set is realized using Def. (4.5). This procedure first gets all the votes from the voter trees and then gets the proposer-set for each level from the genesis to the last level for which the proposer-set can be realized (lines:5-9). It then takes the outer product of these proposer-sets and enumerates many proposer block sequences (line:11). Note that by design, one of these sequences will be the leader block sequence in the future. It then builds a ledger for each proposer block sequence and confirms the transaction if it is present in *all* of the ledgers (lines:12-14).

GetOrderedConfirmedTxs(): First obtain a leader block for each level on proposer blocktree from genesis up until the level which has a confirmed leader block (line:40). Then return the ledger built from these leader blocks.

[6]In absence of an active attack, it will be fast, as described in Section 4.

4 ANALYSIS

In this section, we analyze three aspects of Prism: security, throughput, and latency. Before listing the formal guarantees satisfied by Prism, we first describe at an intuitive level why Prism is able to achieve good latency without sacrificing security.

4.1 Intuition and Sketch of Proofs

In the longest-chain protocol, for a fixed block size and network, the maximum tolerable adversarial hash power β is governed by the block production rate; the faster one produces blocks, the smaller the tolerable β [9, 18]. In Prism, we need to be able to tolerate β adversarial hash power in each of the voter trees and and the proposer tree. Hence, following the observations of [9, 18] each of these trees individually must operate at the same rate as a single longest-chain blocktree in Bitcoin in order to be secure.

The security of Prism is provided by the voter trees; a proposer block is confirmed by votes which are on the longest chains of these voter trees. Consider a conservative confirmation policy for Prism, where we wait for each vote on each voter tree to reach a confirmation reliability $1 - \varepsilon$ before counting it. This would require us to wait for each vote to reach a depth of $k(\varepsilon)$ in its respective tree, where $k(\varepsilon)$ denotes the confirmation depth for reliability $1 - \varepsilon$. This conservative confirmation rule immediately implies that Prism has the same security guarantee as that of each of the voter tree, i.e. that of Bitcoin. However, this rule has as poor a latency as Bitcoin's. For example, for $\varepsilon = 10^{-3}$ and the tolerable adversary power $\beta = 0.3$, the vote has to be 24 blocks deep [16]. With a more intelligent transaction confirmation rule, we can do far better. The key insight is that even though each vote individually stabilizes at the same rate at Bitcoin (i.e., slowly), the *aggregate* opinion can converge much faster because there are many voter trees.

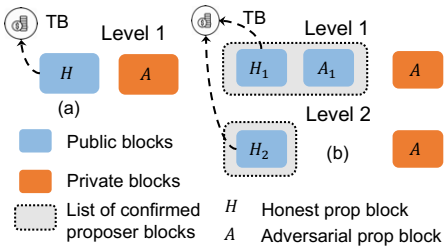

Figure 6: (a) Transaction block is referred to by an isolated honest proposer block. (b) Transaction block is referred to by a non-isolated proposer block but on the next level there is an isolated proposer block. Note that the link from H_2 to TB is implicit; since H_2 is honest, it refers to all unreferenced transaction and proposer blocks, i.e., H_1 and A_1. Since H_1 refers TB, H_2 implicitly does too (Section 3.2)

.

4.1.1 Case 1: Isolated Proposer Block

Consider first the situation when a transaction block TB is referred to by a honest proposer block H which is currently isolated at its level, i.e. no other public proposal block exists at the same level for a certain fixed number of rounds. See Figure 6(a). This case is quite common since the mining rate of the proposer blocks is chosen such that there is little forking in the proposer tree. Block H will

start collecting votes, each of which is on the longest chain of its respective voter tree. Over time, each of these votes will become deeper in its voter chain. An attack by the adversary is to mine a private proposal block A at the same level, and on each of the voter trees fork off and mine a private alternate chain and send its vote to the block A. After leader block H is confirmed, the adversary continues to mine on each of the voter alternate chains to attempt to overtake the public longest chain and shift the vote from H to A. If the adversary can thereby get more votes on A than on H, then its attack is successful.

This can be viewed as the m-chain analog to Nakamoto's private attack on a single chain [16], where instead of having one race between the honest chain and the attack chain we have m such races. In fact, Nakamoto's calculations on the success probability of an attack on a single chain can help us determine how deep we need to wait for the votes to become to confirm the proposer block H. At tolerable adversary power $\beta = 0.3$, the reversal probability in a single chain is 0.45 when a block is 2-deep [16]. With $m = 1000$ voter chains and each vote being 2-deep, the expected number of chains that can be reversed by the adversary is 450. The probability that the adversary got lucky and can reverse more than half the votes, i.e. 500, is about 10^{-3}. Hence to achieve $\varepsilon = 10^{-3}$, we only need to wait for 1000 votes each 2-deep. This incurs much shorter latency than the 24 block depth needed for *each* vote to be reversed with probability 10^{-3}. This reduction in latency is conceptually similar to averaging many unreliable classifiers to form a strong aggregate classifier: the more voter chains there are, the less certainty of permanence each individual vote needs to be, thereby reducing confirmation time. This gain comes without sacrificing security: each voter chain is operating slowly enough to tolerate β adversarial hash power.

Just like Nakamoto's private attack, the attack considered here is a particular attack. Our formal security analysis, sketched in Section 4.1.3, consider all possible attacks in the model. In particular, the attacker can correlate its actions on the different voter chains. However, the confirmation latency behaves similarly to the latency under this attack.

4.1.2 Case 2: Non-isolated Proposer Block

Consider now the case when the the transaction block TB is referred to by a honest proposal block H_1 which is not isolated at its level, i.e. H_1 is matched by an adversarial public proposer block A_1 (the competing proposer block could also be honest). This matching could persist for L levels until reaching a level when there is an isolated honest proposal block. See Figure 6(b) for the special case of $L = 1$. Let us separately consider the life cycle of an honest transaction vs. a double-spent one.

Honest Transaction: A naive approach for confirming TB would be to wait until we can definitively confirm H_1 or A_1. However, this may be slow because of adversarial attacks that try to balance votes. A key insight is that for honest (non-double-spent) transactions, we do not need to know *which* of H_1 and A_1 is confirmed—only that one of them will be confirmed. This weaker form of *list confirmation* works because if A_1 eventually gets confirmed, a later honest proposer block can still refer to H_1 and include TB (Section 3.2). To confirm an honest transaction at level i, we need two events: (1) list confirmation of all levels up to i; (2) an isolated honest proposer at level i. Once we have list-confirmed a set of proposer blocks at level i referring TB (e.g., either H_1 or A_1 will be the leader), we know that no other block can be the leader at that level. However, list confirmation alone is not enough for honest transaction confirmation if the transaction is not present in all ledgers. In that case, we also need to wait for an isolated honest proposer level, where the proposer block will implicitly or explicitly include TB in the ledger. Once this isolated honest proposer level is confirmed *and* all the preceding levels are list-confirmed, we can be sure that TB will appear in the final ledger. The confirmation latency is thus the maximum of two parts:

(1) List confirmation. We fast confirm that the adversary cannot produce a private block A with more votes than the votes of public blocks H_1 and A_1. The logic is similar to the case of isolated honest proposer block discussed above, viewing the situation as a race between honest nodes voting for the public blocks H_1 or A_1 and adversary voting for A. Adversarial actions (e.g., presenting first H_1 to half the honest nodes and A_1 to the other half) can cause the number of votes to be evenly split between H_1 and A_1, which can slow down list confirmation, albeit not significantly.

(2) Isolated honest proposer level. In Figure 6(a), if we wait until level 2, we see an isolated public proposer block H_2 which can be fast confirmed (Section 4.1.1). At this point, we know that the final leader sequence at levels $1, 2$ is either H_1, H_2 or A_1, H_2, both of which contain our honest transaction since H_2 refers to all previous unreferred proposer blocks. Since isolated honest proposer blocks happen frequently (Section 4.1.3), this step is fast.

Double-Spent Transaction: To confirm double-spent transactions, we need stronger conditions than those listed above: namely, instead of list confirmation, we need *unique block confirmation*, confirming which block at a proposer level will be the ultimate leader. This is achieved once list confirmation occurs *and* one of the list-confirmed blocks can be reliably declared the winner. If one of the public proposer blocks H_1 or A_1 gathers many more votes than the other block, then we can fast confirm a unique leader, even for double-spent transactions; this happens both in the absence of active attacks and under some classes of attacks (Section 5). However, other adversarial attacks (such as balancing the votes on H_1 and A_1) can cause the number of votes to be evenly split between H_1 and A_1, so we cannot fast confirm a leader block. In this case, we must wait until every vote on H_1 and A_1 stabilizes, in which case either H_1 or A_1 is confirmed and only one of the double-spent transactions is accepted. A content-dependent tie breaking rule can be used to break ties after votes are stabilized.

4.1.3 Sketch of Security and Latency Proofs

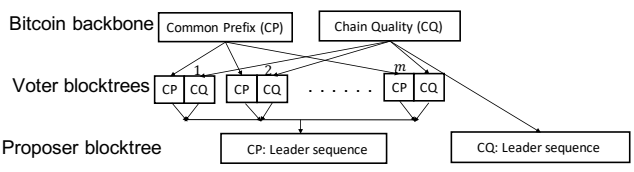

Figure 7: Common-prefix and chain-quality properties of voter chains imply common-prefix and chain-quality properties of the proposer leader sequence.

We next provide a sketch of the proof structure. Due to space constraints, the outline of the proofs are presented in Appendices D, E, and F; all full proofs can be found in our extended technical report [3]. To translate the above intuitive arguments into formal security and latency proofs, we borrow key ideas from [9], but also require several new insights. [9] proves the consistency and liveness of the Bitcoin backbone protocol by first proving two key properties: common-prefix and chain-quality. Similarly, to show that Prism achieves consistency and liveness, we need to show that the proposer leader sequence satisfies these properties. The results of [9] do not directly apply because the proposer leader sequence is not determined by a single longest-chain protocol; rather, it is determined by a combination of the proposer tree and the aggregate voter tree votes. As shown in Figure 7, we prove the two properties for the proposer and the voter trees and use them to prove corresponding properties for the leader sequence. Specifically:

(1) Each voter tree is constructed according to the backbone protocol, and hence satisfies the chain-quality and common-prefix property. Chain-quality of the voter trees implies that honest nodes will continually be able to vote for proposer blocks from every voter tree and at every proposer level . Common-prefix implies that all these votes will eventually stabilize. This further implies that the leader sequence satisfies the common-prefix property (Theorem 4.1), since the leader block at each level will eventually stabilize. Hence, the resulting ledger is consistent. The leader-sequence also can be shown to have a certain chain quality (Lemma D.6) and this ensures liveness of the ledger (Theorem 4.2).

(2) To show fast confirmation of all honest transactions, we follow the intuitive arguments above. We first show that an isolated proposer block, or an honest proposer block that does not have a competing adversarial proposer block for a certain duration of time, appears in constant expected time (independent of ε). Specifically, the honest users are mining proposer blocks at the rate $(1-\beta)\bar{f}_p$ whereas the adversary is mining at rate $\beta\bar{f}_p$. Since $\beta < 0.5$, the adversary is mining slower than the honest users, and within the next $\frac{1}{1-2\beta}$ levels in expectation, there is a level on which the adversary cannot *immediately* create a competing block with the honest block [7]. Similarly, an isolated level on which the adversary cannot match the honest block for next R rounds after the honest block is mined happens within $\frac{1+2R\bar{f}_v}{1-2\beta}$ levels in expectation.

(3) We next show that we can fast confirm an isolated public honest proposer block. The argument has two parts: i) the isolated honest block wins enough votes; 2) the leader block persists, i.e., wins the vote race against a private adversarial proposer block for all time. The first part follows from the chain-quality of the voter chains, which ensures that there is a steady stream of honest votes for the public proposer block until it gathers a sufficiently large fraction of total votes (Lemma E.7). The second part follows from common-prefix of the voter trees, which ensures that a large fraction of votes cannot be reversed by the adversary (Lemma E.9).

(4) Fast *list* confirmation of proposer blocks at all previous levels can be proved similarly (see Lemma E.10 and Theorem 4.6). Now, Prism ensures that at each proposer level, one of the list-confirmed blocks will remain in the ledger. This, combined with the assurance that every transaction will be either directly or indirectly referred by the isolated proposal block, ensures that all honest transactions are entered into the ledger. This lets Prism confirm honest transactions within a short time (see Theorem 4.7).

Note that [9] proves the k-common-prefix property is satisfied with high probability only for large k. Similarly, chain-quality is shown to be satisfied with high probability only over a large number of blocks. While this is sufficient to prove (1) and (2) above for the consistency and liveness of the eventual ledger, it is not sufficient to prove (4) and (5) for fast confirmation latency, since we need these two properties over short latencies, i.e. windows of few blocks. In these small time windows, these properties do not hold with high probability *microscopically*, for every *individual* voter tree. However, since the proposer leader block depends only on the *macroscopic* vote counts, we only need to show that these properties hold with high probability macroscopically, for a good fraction of voter trees.

4.2 Parameter Selection

We first specify the parameters of Prism in terms of the parameters of the physical network. First, recall that the network delay of a block containing B transactions is given by $\Delta = \frac{B}{C} + D$. Let B_t, B_v, and B_p be the size of transaction, voter, and proposer blocks respectively, in units of number of transactions. The network delays Δ_t, Δ_v, and Δ_p for each type of block are thus given by:

$$\Delta_t = \frac{B_t}{C} + D, \quad \Delta_v = \frac{B_v}{C} + D, \quad \Delta_p = \frac{B_p}{C} + D. \tag{9}$$

Given that different block types have different sizes and network delays, what is a reasonable choice for Δ, the duration of a round? Since the synchronous model is used for security analysis, and the security of Prism depends only on the network delay of the proposer and voter blocks but not of the transaction blocks, we choose: $\Delta = \max\{\Delta_p, \Delta_v\}$. Moreover, the voter blocks and the proposer blocks contain only reference links and no transactions, so their sizes are expected to be small. Assuming the bandwidth-delay product $CD/\max\{B_v, B_p\} \gg 1$, we have that the network delay $\Delta = \max\{\frac{B_v}{C}, \frac{B_p}{C}\} + D \approx D$, the smallest possible.

To provide security, we set the mining rate $\bar{f}_v := f_v D$ on each voter tree such that each voter tree is secure under the longest chain rule. According to [9] it should satisfy

$$\bar{f}_v < \frac{1}{1-\beta} \log \frac{1-\beta}{\beta}. \tag{10}$$

We also set the proposer and voter mining rates to be the same, i.e. $f_p = f_v$. This is not necessary but simplifies the notation.

Third, to utilize 90% of the communication bandwidth for carrying transaction blocks, we set $f_t B_t = 0.9C$. The individual choices of f_t and B_t are not very important, but choosing large B_t and small f_t is preferable to ensure that the number of reference links to transaction blocks per proposer block is small, thus giving a small proposer block size B_p.

Finally, speed up voting, we maximize the number of voter chains subject to the *stability constraint* of Sec. 2: $f_p B_p + m f_v B_v + f_t B_t < C$. Substituting the values of f_v, f_p and $f_t B_t$, we get

$$m = \frac{0.1CD}{\bar{f}_v B_v} - \frac{B_p}{B_v} \geq \frac{(1-\beta)}{\log(\frac{1-\beta}{\beta})} \cdot \frac{CD}{B_v} - \frac{B_p}{B_v}$$

[7] Random walk analysis

i.e. the number of voter trees is at least proportional to CD/B_v, the bandwidth-delay product in unit of voting blocks. This number is expected to be large, which is a key advantage. The only degree of freedom left is the choice of $\bar{f}_v$, subject to (10). We will return to this issue in Section 4.5 when we discuss fast confirmation latency.

4.3 Total Ordering

In this subsection, we show that Prism can achieve total transaction ordering for any $\beta < 0.5$ using the procedure GetOrderedConfirmedTxs() in Algorithm 2. That is, as long as the adversary's hash power is less than 50%, transactions can be ordered with consistency and liveness guarantees. Following [9], we do so by first establishing two backbone properties: common-prefix and chain quality of the *proposer leader sequence*. Let $\mathcal{P}(r)$ denote the set of proposer blocks mined by round r. Let $\mathcal{P}_\ell(r) \subseteq \mathcal{P}(r)$ denote the set of proposer blocks mined on level ℓ by round r. Let the first proposer block on level ℓ be mined in round R_ℓ. Let $V_p(r)$ denote the number of votes on proposer block $p \in \mathcal{P}(r)$ at round r. Recall that only votes from the main chains of the voter trees are counted. The *leader block* on level ℓ at round r, denoted by $p^*_\ell(r)$, is the proposer block with maximum number of votes in the set $\mathcal{P}_\ell(r)$ i.e, $p^*_\ell(r) := \underset{p\in\mathcal{P}_\ell(r)}{\text{argmax}}\ V_p(r)$, where tie-breaking is done in a hash-dependent way. The *leader sequence* up to level ℓ at round r, denoted by $\mathtt{LedSeq}_\ell(r)$ is:

$$\mathtt{LedSeq}_\ell(r) := [p^*_1(r), p^*_2(r), \cdots, p^*_\ell(r)]. \tag{11}$$

The leader sequence at the end of round $r_{\max}$, the end of the horizon, is the *final leader sequence*, $\mathtt{LedSeq}_\ell(r_{\max})$.

Theorem 4.1 (Leader sequence common-prefix property). *Suppose $\beta < 0.5$. For a fixed level ℓ, we have*

$$\mathtt{LedSeq}_\ell(r) = \mathtt{LedSeq}_\ell(r_{\max}) \quad \forall r \geq R_\ell + r(\varepsilon) \tag{12}$$

with probability $1-\varepsilon$, where $r(\varepsilon) = \frac{1024}{\bar{f}_v(1-2\beta)^3}\log\frac{8mr_{\max}}{\varepsilon}$, and R_ℓ is the round in which the first proposer block on level ℓ was mined.

Proof. See Appendix D. □

Theorem 4.2 (Liveness). *Assume $\beta < 0.5$. Once a transaction enters into a transaction block, w.p $1-\varepsilon$ it will eventually be pointed to by a permanent leader sequence block after*

$$O\left(\frac{\log(1/\varepsilon)}{(1-2\beta)^4}\right) \quad \textit{rounds.}$$

Proof. See Appendix D. □

Theorems 4.1 and 4.2 yield the following:

Theorem 4.3. *The ledger has consistency and liveness with the expected ε-*latency *for all transactions (Def. 8) to be at most $O(\log(1/\varepsilon))$ for $\beta < 0.5$.*

4.4 Throughput

We now analyze the transaction throughput of Prism. The leader sequence blocks of Prism orders all the transactions in the transaction blocks they refer to. Due to liveness, all transaction blocks are referred to by some proposer blocks. Since the transaction block generation rate $f_t B_t$ is chosen to be $0.9C$ transactions per second, assuming a worst case that only honest blocks carry transactions yield a throughput of $0.9(1-\beta)C$ transactions per seconds.

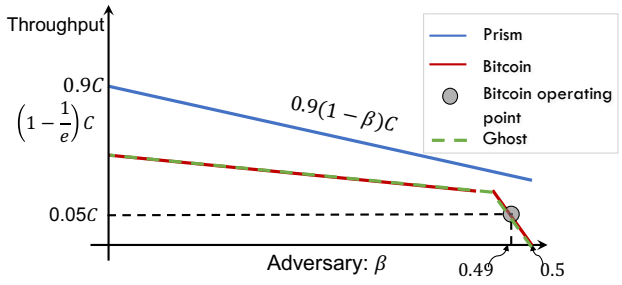

Figure 8: Throughput versus β tradeoffs of Prism, Bitcoin **and** GHOST**. The tradeoffs for the baseline protocols are upper bounds, while that for** Prism **is exact.**

This seems to give the advertised goal, but there is a catch: blocks mined in the same round may contain the same transactions, since transactions are broadcast to the entire network. To achieve the full throughput, one can minimize the transaction redundancy in the blocks by **scheduling** different transactions to different blocks. Concretely, we split the transactions randomly into q queues, and each honest block is created from transactions drawn from one randomly chosen queue. Thinking of each transaction queue as a color, we have transaction blocks of q different colors.

We will only have honest blocks with redundant transactions if two or more blocks of the same color are mined in the same round. The number of honest blocks of the same color mined at the same round is distributed as Poisson with mean $(1-\beta)f_t\Delta/q$, and so the throughput of non-redundant blocks of a given color is the probability that at least one such block is mined in a round, i.e. $1-e^{-(1-\beta)f_t\Delta/q}$ blocks per round. The total throughput of non-redundant honest blocks of all colors is

$$q\left[1-e^{-(1-\beta)f_t\Delta/q}\right] \text{ blocks per round.} \tag{13}$$

For large q, this approaches $(1-\beta)f_t\Delta$ blocks per round, which equals $0.9(1-\beta)C$ transactions per second when we set $f_t = 0.9C/B_t$. Thus, we achieve the claimed result (4).

Theorem 4.4 (Throughput). *Theorem 4.8 guarantees that all the transactions proposed before round $r - O(\log(1/\varepsilon))$ are confirmed by round r. Therefore* Prism *can support $\lambda = 0.9(1-\beta)C$ throughput (Def. 7), where $U_\varepsilon = O(\log(1/\varepsilon))$.*

Note that the throughput of Prism as a fraction of capacity does not vanish as $\beta \to 0.5$, unlike Bitcoin (Figure 8). Indeed, in order for Bitcoin to be secured against Nakamoto's private attack [16] in that regime, it is necessary that $f\Delta$, the expected number of blocks mined per network delay round, approaches 0 so that very little forking occurs among the honest nodes and the honest nodes can grow the longest chain faster than the adversary. Note that for a given block size B, the throughput is bounded by:

$$fB = f\Delta \cdot B/\Delta = f\Delta \cdot B/(B/C + D) < f\Delta C \text{ tx/second}$$

Hence, in the regime where $\beta \to 0.5$, Bitcoin can only achieve a *vanishing* fraction of the network capacity. Because the mining rate of GHOST is similarly security-limited, its throughput has similar behavior as Bitcoin (see [12] and Appendix I in [3]).

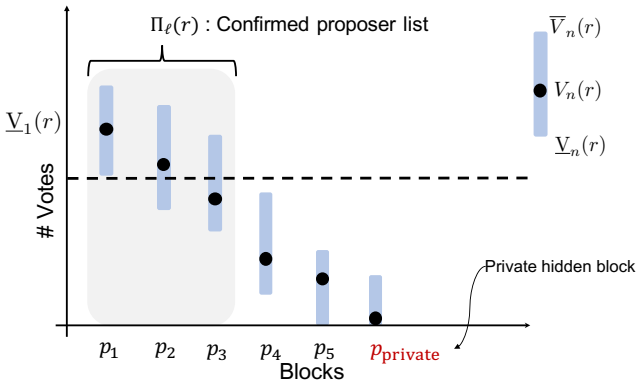

Figure 9: Public proposer block p_1 has the largest lower confidence bound, which is larger than the upper confidence bound of the private block. So list confirmation is possible and the set confirmed is $\Pi_\ell(r) = \{p_1, p_2, p_3\}$.

4.5 Fast confirmation latency

4.5.1 List confirmation latency

We convert the intuition from Section 4.1 to a formal rule for fast confirming a *set* of proposer blocks, which enables confirming a list of proposer sequences. The idea is to have *confidence intervals* around the number of votes cast on each proposer block. Figure 9 gives an example where there are 5 proposal blocks in public at a given level, and we are currently at round r. The confidence interval $[\underline{V}_n(r), \overline{V}_n(r)]$ for the votes on proposer block p_n bounds the maximum number of votes the block can lose or gain from uncast votes and votes reversed by the adversary. We also consider a potential private proposer block, with an upper bound on the maximum number of votes it can accumulate in the future. We can fast confirm a set of proposal blocks whenever the upper confidence bound of the private block is below the lower confidence bound of the public proposal block with the largest lower confidence bound.

More formally: As defined earlier, $\mathcal{P}_\ell(r) = \{p_1, p_2 \ldots\}$ is the set of proposer blocks at level ℓ at round r. Let $V_n^d(r)$ be the number of votes at depth d or greater for proposer block p_n at round r. Define:

$$\delta_d := \max\left(\frac{1}{4\bar{f}_v d}, \frac{1-2\beta}{8\log m}\right), \varepsilon' = 1 - r_{\max}^2 e^{-\frac{(1-2\beta)m}{16\log m}},$$

$$\underline{V}_n(r) := \max_{d\geq 0}\left(V_n^d(r) - 2\delta_d m\right)_+,$$

$$\overline{V}_n(r) := m - \sum_{p_{n'}\in\mathcal{P}_\ell(r)\setminus\{p_n\}} \underline{V}_{n'}(r),$$

$$\underline{V}_{\text{private}}(r) := 0, \quad \overline{V}_{\text{private}}(r) := m - \sum_{p_{n'}\in\mathcal{P}_\ell(r)} \underline{V}_{n'}(r).$$

Definition 4.5. **Proposer set confirmation policy**: *If*

$$\max_n \underline{V}_n(r) > \overline{V}_{private}(r), \tag{14}$$

then we confirm the the set of proposer blocks $\Pi_\ell(r)$, where

$$\Pi_\ell(r) := \{p_n : \overline{V}_n(r) > \max_i \underline{V}_i(r)\}. \tag{15}$$

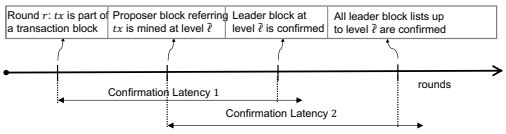

Figure 10: Components of the latency: a) Confirmation latency 1 is analyzed in Theorem 4.7, and b) Confirmation latency 2 is analyzed in Theorem 4.6.

Next, we show that we can confirm proposer sets up to level ℓ with an expected latency independent of ε, and the final leader sequence is contained in the outer product of the confirmed sets.

Theorem 4.6 (List common-prefix property). *Suppose $\beta < 0.5$. Suppose the first proposer block at level ℓ appears at round R_ℓ. Then w.p. ε', we can confirm proposer sets $\Pi_1(r)), \ldots, \Pi_\ell(r)$ for all rounds $r \geq R_\ell + R_\ell^{conf}$, where*

$$\mathbb{E}[R_\ell^{conf}] \leq \frac{2808}{(1-2\beta)^3\bar{f}_v}\log\frac{50}{(1-2\beta)} + \frac{256}{(1-2\beta)^6\bar{f}_v m^2}, \tag{16}$$

and $p^*_{\ell'}(r_{\max}) \in \Pi_{\ell'}(r) \quad \forall \ell' \leq \ell$ and $r \geq R_\ell + R_\ell^{conf}$.

Proof. See Appendix E. □

Let us express the latency bound (16) in terms of physical parameters. If we set the voting rate $\bar{f}_v$ equal to the largest possible given the security constraint (10): $\bar{f}_v = \frac{1}{1-\beta}\log\frac{1-\beta}{\beta}$, then according to (11), we have

$$m = \frac{0.1(1-\beta)}{\log(\frac{1-\beta}{\beta})}\cdot\frac{CD}{B_v} - \frac{B_p}{B_v}.$$

With this choice of parameters, and in the regime where the bandwidth-delay product CD/B_v is large so that the second term in (16) can be neglected, the expected latency for list confirmation is bounded by $c_1(\beta)D$ seconds, i.e. proportional to the propagation delay. Here,

$$c_1(\beta) := \frac{2808(1-\beta)}{(1-2\beta)^3\log\frac{1-\beta}{\beta}}\log\frac{50}{(1-2\beta)}$$

and is positive for $\beta < 0.5$. The confirmation error probability is exponentially small in CD/B_v. This is the constant part of the latency versus security parameter tradeoff of Prism in Fig. 1. Since CD/B_v is very large in typical networks, a confirmation error probability exponentially small in CD/B_v is already very small. To achieve an even smaller error probability ε we can reduce the voting rate $\bar{f}_v$ smaller below the security constraint (10) and increase the number of voter chains. More specifically, we set

$$\bar{f}_v = \frac{0.1CD}{B_v\log\frac{1}{\varepsilon}}, \tag{17}$$

resulting in $m = \log\frac{1}{\varepsilon} - \frac{B_p}{B_v} \approx \log\frac{1}{\varepsilon}$, yielding the desired security parameter. Again neglecting the second term in (16), the corresponding latency bound is

$$\frac{c_2(\beta)B_v}{C}\log\frac{1}{\varepsilon} \quad \text{seconds},$$

where $c_2(\beta) := \frac{54000}{(1-2\beta)^3}\log\frac{50}{(1-2\beta)}$. This is the linearly increasing part of the Prism curve in Figure 1, with slope inversely proportional to the network capacity C/B_v.

4.5.2 Fast confirmation of honest transactions

In the previous subsection we have shown that one can fast confirm a set of proposer block sequences which is guaranteed to contain the prefix of the final totally ordered leader sequence. As discussed in Section 3, each of these proposer block sequence creates an ordered ledger of transactions using the reference links to the transaction blocks. In each of these ledgers, double-spends are removed to sanitize the ledger. If a transaction appears in *all* of the sanitized ledgers in the list, then it is guaranteed to be in the final total ordered sanitized ledger, and the transaction can be fast confirmed. All honest transactions without double-spends eventually have this *list-liveness* property; when only a single honest proposer block appears in a level and becomes the leader, it adds any honest transactions that have not already appeared in at least one ledger in the list. Due to the positive chain-quality of the leader sequence (Theorem 4.2), an isolated honest level eventually occurs. The latency of confirming honest transactions is therefore bounded by the sum of the latency of list confirmation in Theorem 4.6 plus the latency of waiting for this event to occur (Fig. 10). The latter is given by:

Theorem 4.7 (List-liveness). *Assume $\beta < 0.5$. If an honest transaction without double spends is mined in a transaction block in round r, then w.p. $1 - r_{\max}^2 e^{-\frac{m}{16 \log m}}$ it will appear in all of the ledgers corresponding to proposer block sequences after an expected latency no more than*

$$\frac{2592}{(1-2\beta)^3 \bar{f}_v} \log \frac{50}{(1-2\beta)} \quad \textit{rounds.}$$

Proof. See Appendix F. □

Figure 10 shows the various components of the overall latency we analyzed. We can see that the confirmation latency from the time an honest transaction enters a blocks to the time it is confirmed is bounded by the sum of the latencies in Theorem 4.6 and 4.7.

Repeating the analysis of Thm. 4.3, we get the following:

Theorem 4.8 (Latency). *Theorems 4.6 and 4.7 guarantee that the expected ε-*latency *for all **honest** transactions (Def. 8) is at most $r(\beta)$ rounds for $\beta < 0.5$, where*

$$r(\beta) := \max\left\{c_1(\beta), c_2(\beta)\frac{B_v}{DC}\log\frac{1}{\varepsilon}\right\},$$

where

$$c_1(\beta) := \frac{5400(1-\beta)}{(1-2\beta)^3 \log\frac{1-\beta}{\beta}} \log\frac{50}{(1-2\beta)}$$

$$c_2(\beta) := \frac{54000}{(1-2\beta)^3} \log\frac{50}{(1-2\beta)},$$

Therefore the honest transactions are confirmed in

$$\max\left\{c_1(\beta)D,\ c_2(\beta)\frac{B_v}{C}\log\frac{1}{\varepsilon}\right\} \quad \text{seconds.}$$

5 SIMULATIONS

Theorem 4.8 provides a theoretical upper bound on the expected latency, which matches the physical limit of propagation time up to constant factors. Characterizing the exact constants is an interesting research direction, but outside the scope of this paper. On the other hand, one can empirically estimate the average latency values by simulating the Prism protocol and its confirmation rule. The purpose of this section is to conduct such a simulation in the honest setting as well as a variety of adversarial settings.

Setup. We simulate a network with $m = 1,000$ voter chains, in which $D \approx \Delta = 1$ sec. We run our proposer tree and each voter tree at a rate of $\bar{f} = 1$ block /10 sec. Our simulations measure the latency for transaction confirmation under three scenarios: no attack, a balancing attack, and a censorship attack. By design, our confirmation rule is simultaneously robust against the common private Nakamoto attack [16], where the adversary withholds a proposer block as well as corresponding forked voter blocks in order to reverse a confirmed proposal block. In this section, we show figures for an adversary deploying $\tilde{\beta} = 0.25$ fraction of total hash power, where $\tilde{\beta}$ denotes the fraction of hash power being actually used for the attack (whereas β is the maximum tolerable fraction of adversarial hash power, without losing consistency and liveness). We set the confirmation reliability conservatively at $\varepsilon = e^{-20}$. Experiments for additional parameter settings can be found in the Appendix in [3]. We compare against the longest-chain protocol, for the same block generation rate of 1 block per 10 seconds.

No Attack. We start by considering a setting where Prism's parameters are chosen to withstand an attacker of hash power β, but the adversary is not actively conducting any attack. Since the confirmation rule must still defend against β adversarial hash power, latency depends on β. Honest nodes vote on the earliest-seen proposer block, with results shown in Figure 11(a). In Bitcoin, a confirmed transaction has to be deeper in the chain for larger β; in Prism, the voter blocks have to be deeper. We see that Prism's latency is significantly smaller than that of Nakamoto's longest chain protocol, and much closer to the physical limit. Note that since there is no active adversary, double-spend transactions can be resolved with the same latency as honest transactions.

Balancing Attack. In a balancing attack, the goal of the adversary is to prevent confirmation by casting all of its votes so as to compete with the current proposer leader block. We begin this attack with two competing proposer blocks at the same level (say level 0), A and B. Consider an honest (non-double-spent) transaction that is referred by at least one of the two proposer blocks. The adversary's goal is to prevent the system from confirming this transaction by balancing votes on the two proposer blocks. That is, if block A currently has the majority of votes and the adversary mines a voter block in the ith voter tree: (1) If voter tree i has not yet voted on level 0, the adversary votes on the minority block, B. (2) If voter tree i voted on level 0 for block B, the adversary appends its block to the longest chain, thereby reinforcing the vote for the losing proposer block. (3) If voter tree i voted on level 0 for block A, the adversary tries to fork the ith voter tree to vote for B instead. If there is no vote for B in the voter tree, the adversary creates one. If there is already a fork voting for B, the adversary appends to this fork. The balancing attack is one of the most severe and natural attacks on Prism. The results of this simulation are shown in Figure 11(b). Notice that the latency of honest transaction confirmation increases by a factor of about 2x under a balancing attack, but does not affect the longest-chain protocol. Despite this, Prism's latency is still far lower than that of the longest-chain protocol.

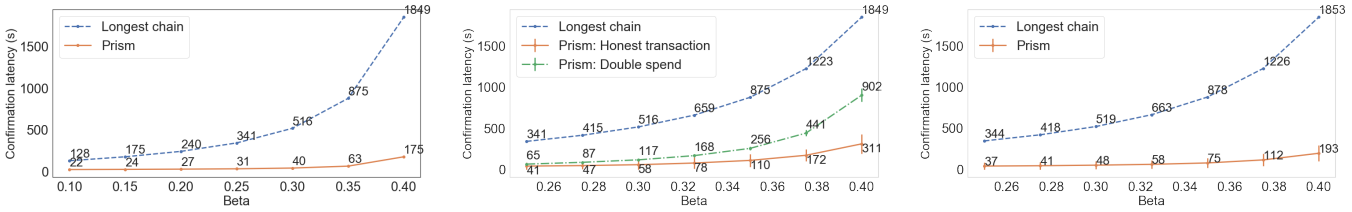

Figure 11: (a) Confirmation latency of honest transactions with no attack. The x-axis denotes the maximum tolerable fraction of adversarial hash power β. (b) Transaction latency in the presence of an adversarial balancing attack from $\tilde{\beta} = 0.25$ active hash power, for honest and double-spent transactions. (c) Confirmation latency under a censorship attack with $\tilde{\beta} = 0.25$ hash power. Honest and double-spent transactions have the same latency, both for Prism and for longest chain.

Next, we consider double-spent transactions. The latency for double-spent transactions is the same as honest transactions in the longest-chain protocol, so the blue curve does not change. However, the double-spent transaction latency for Prism grows substantially, approaching that of the longest-chain protocol. Indeed, as the active $\tilde{\beta}$ fraction approaches 0.5, Prism's latency on double-spent transactions in the presence of attacks on the confirmation process actually exceeds that of the longest-chain protocol, as discussed in Section 4.1.

Censorship Attack Finally, we consider an attacker whose goal is simply to slow down the confirmation of blocks by proposing empty proposer and voter blocks. This has two effects: (1) it delays the creation of a proposer block referencing the transaction block containing the transaction, and (2) it delays the confirmation of such a proposer block by delaying the creation of votes on the proposer tree. The results of this attack are shown in Figure 11(c). The censorship attack adds a delay of between 15-20 seconds to Prism's confirmation delay compared to the non-adversarial setting. The effect is smaller for the longest-chain protocol, since the only delay comes from delaying the insertion of a transaction into a block. Under a censorship attack, double-spent transactions have the same latency as honest ones.

ACKNOWLEDGEMENT

We thank the Distributed Technologies Research Foundation, the Army Research Office under grant W911NF-18-1-0332-(73198-NS), the National Science Foundation under grants 1705007 and 1651236 for supporting their research program on blockchain technologies. We thank Applied Protocol Research Inc. for support and for providing a conducive environment that fostered this collaborative research. We also thank Andrew Miller and Mohammad Alizadeh for their comments on an earlier draft. We also thank Soubhik Deb for helping us with the simulations.

REFERENCES

[1] I. Abraham, D. Malkhi, K. Nayak, L. Ren, and A. Spiegelman. Solida: A blockchain protocol based on reconfigurable byzantine consensus. *arXiv preprint arXiv:1612.02916*, 2016.
[2] Gavin Andresen. Weak block thoughts. bitcoin-dev. https://lists.linuxfoundation.org/pipermail/bitcoin-dev/2015-September/011157.html.
[3] Vivek Bagaria, Sreeram Kannan, David Tse, Giulia Fanti, and Pramod Viswanath. Prism: Deconstructing the blockchain to approach physical limits. *arXiv preprint*, 2019. Extended version.
[4] Vitalik Buterin. On slow and fast block times, 2015. https://blog.ethereum.org/2015/09/14/on-slow-and-fast-block-times/.
[5] C. Decker and R. Wattenhofer. Information propagation in the bitcoin network. In *IEEE P2P 2013 Proceedings*, pages 1–10, Sept 2013.
[6] Christian Decker, Jochen Seidel, and Roger Wattenhofer. Bitcoin meets strong consistency. In *Proceedings of the 17th International Conference on Distributed Computing and Networking*, page 13. ACM, 2016.
[7] Ittay Eyal, Adem Efe Gencer, Emin Gün Sirer, and Robbert Van Renesse. Bitcoin-ng: A scalable blockchain protocol. In *NSDI*, pages 45–59, 2016.
[8] Matthias Fitzi, Peter Gaži, Aggelos Kiayias, and Alexander Russell. Parallel chains: Improving throughput and latency of blockchain protocols via parallel composition. Cryptology ePrint Archive, Report 1119, 2018.
[9] Juan Garay, Aggelos Kiayias, and Nikos Leonardos. The bitcoin backbone protocol: Analysis and applications. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 281–310. Springer, 2015.
[10] Aggelos K. and Giorgos P. On trees, chains and fast transactions in the blockchain. 2016.
[11] Dina Katabi, Mark Handley, and Charlie Rohrs. Congestion control for high bandwidth-delay product networks. *ACM SIGCOMM computer communication review*, 32(4):89–102, 2002.
[12] Lucianna Kiffer, Rajmohan Rajaraman, et al. A better method to analyze blockchain consistency. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, pages 729–744. ACM, 2018.
[13] Eleftherios Kogias, Philipp Jovanovic, Nicolas Gailly, Ismail Khoffi, Linus Gasser, and Bryan Ford. Enhancing bitcoin security and performance with strong consistency via collective signing. In *25th USENIX Security Symposium*, 2016.
[14] Yoad Lewenberg, Yonatan Sompolinsky, and Aviv Zohar. Inclusive block chain protocols. In *International Conference on Financial Cryptography and Data Security*, pages 528–547. Springer, 2015.
[15] Chenxing Li, Peilun Li, Wei Xu, Fan Long, and Andrew Chi-chih Yao. Scaling nakamoto consensus to thousands of transactions per second. *arXiv preprint arXiv:1805.03870*, 2018.
[16] Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system. 2008.
[17] Christopher Natoli and Vincent Gramoli. Balance attack against proof-of-work blockchains: The r3 testbed as an example. *arXiv preprint arXiv:1612.09426*, 2016.
[18] R Pass, L Seeman, and A Shelat. Analysis of the blockchain protocol in asynchronous networks. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, 2017.
[19] R. Pass and E. Shi. Fruitchains: A fair blockchain. In *Proceedings of the ACM Symposium on Principles of Distributed Computing*. ACM, 2017.
[20] Rafael Pass and Elaine Shi. Hybrid consensus: Efficient consensus in the permissionless model. In *LIPIcs-Leibniz International Proceedings in Informatics*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2017.
[21] Rafael Pass and Elaine Shi. Thunderella: Blockchains with optimistic instant confirmation. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 2018.
[22] Peter R Rizun. Subchains: A technique to scale bitcoin and improve the user experience. *Ledger*, 1:38–52, 2016.
[23] Y Sompolinsky, Y Lewenberg, and A Zohar. Spectre: A fast and scalable cryptocurrency protocol. *IACR Cryptology ePrint Archive*, 2016:1159.
[24] Y Sompolinsky and A Zohar. Phantom: A scalable blockdag protocol, 2018.
[25] Yonatan Sompolinsky and Aviv Zohar. Secure high-rate transaction processing in bitcoin. In *International Conference on Financial Cryptography and Data Security*, pages 507–527. Springer, 2015.
[26] TierNolan. Decoupling transactions and pow. Bitcoin Forum. https://bitcointalk.org/index.php?topic=179598.0.
[27] Haifeng Yu, Ivica Nikolic, Ruomu Hou, and Prateek Saxena. OHIE: blockchain scaling made simple. *CoRR*, abs/1811.12628, 2018.

A PSEUDOCODE

Algorithm 1 Prism: Mining

```
1: procedure Main( )
2:     Initialize()
3:     while True do
4:         header, Ppf, Cpf = PowMining()
5:         // Block contains header, parent, content and merkle proofs
6:         if header is a tx block then
7:             block ← ⟨header, txParent, txPool, Ppf, Cpf⟩
8:         else if header is a prop block then
9:             block ← ⟨header, prpParent, unRfTxBkPool, Ppf, Cpf⟩
10:        else if header is a block in voter blocktree i then
11:            block ← ⟨header, vtParent[i], votesOnPrpBks[i], Ppf, Cpf⟩
12:        BroadcastMessage(block)                    ▷ Broadcast to peers
13: procedure Initialize( )                           ▷ All variables are global
14:    // Blockchain data structure C = (prpTree, vtTree)
15:    prpTree ← genesisP                             ▷ Proposer Blocktree
16:    for i ← 1 to m do vtTree[i] ← genesisM_i
17:    // Parent blocks to mine on
18:    prpParent ← genesisP                           ▷ Proposer block to mine on
19:    for i ← 1 to m do
20:        vtParent[i] ← genesisM_i                   ▷ Voter tree i block to mine on
21:    // Block content
22:    txPool ← ϕ                                     ▷ Tx block content: Txs to add in tx bks
23:    unRfTxBkPool ← ϕ                               ▷ Prop bk content1: Unreferred tx bks
24:    unRfPrpBkPool ← ϕ                              ▷ Prop bk content2: Unreferred prp bks
25:    for i ← 1 to m do votesOnPrpBks(i) ← ϕ         ▷ Vot. blk cnt.
26: procedure PowMining( )
27:    while True do
28:        txParent ← prpParent
29:        // Assign content for all block types/trees
30:        for i ← 1 to m do vtContent[i] ← votesOnPrpBks[i]
31:        txContent ← txPool
32:        prContent ← (unRfTxBkPool, unRfPrpBkPool)
33:        // Define parents and content Merkle trees
34:        parentMT ←MerklTree(vtParent, txParent, prpParent)
35:        contentMT ←MerklTree(vtContent, txContent, prContent)
36:        nonce ← RandomString(1^κ)
37:        // Header is similar to Bitcoin
38:        header ← ⟨ parentMT.root, contentMT.root, nonce ⟩
39:        // Sortition into different block types/trees
40:        if Hash(header) ≤ mf_v then                ▷ Voter block mined
41:            i ← ⌊Hash(header)/f_v⌋ and break       ▷ on tree i
42:        else if mf_v <Hash(header) ≤ mf_v + f_t then
43:            i ← m + 1 and break                    ▷ Tx block mined
44:        else if mf_v + f_t < Hash(header) ≤ mf_v + f_t + f_p then
45:            i ← m + 2 and break                    ▷ Prop block mined
       // Return header along with Merkle proofs
46:    return ⟨header, parentMT.proof(i), contentMT.proof(i)⟩
47: procedure ReceiveBlock(B)                         ▷ Get block from peers
48:    if B is a valid transaction block then
49:        txPool.removeTxsFrom(B) and unRfTxBkPool.append(B)
50:    else if B is a valid block on i^th voter tree then
51:        vtTree[i].append(B) and vtTree[i].append(B.ancestors())
52:        if B.chainlen > vtParent[i].chainlen then
53:            vtParent[i] ← B and votesOnPrpBks(i).update(B)
54:    else if B is a valid prop block then
55:        if B.level == prpParent.level+1 then
56:            prpParent ← B
57:            for i ← 1 to m do                      ▷ Add vote on level ℓ on all m trees
58:                votesOnPrpBks(i)[B.level] ← B
59:        else if B.level > prpParent.level+1 then
60:            // Miner doesnt have block at level prpParent.level+1
61:            RequestNetwork(B.parent)
62:        prpTree[B.level].append(B), unRfPrpBkPool.append(B)
63:        unRfTxBkPool.removeTxBkRefsFrom(B)
64:        unRfPrpBkPool.removePrpBkRefsFrom(B) '
```

Algorithm 2 Prism: Tx confirmation

```
1: procedure IsTxConfirmed(tx)
2:     Π ← ϕ                                          ▷ Array of set of proposer blocks
3:     for ℓ ← 1 to prpTree.maxLevel do
4:         votesNdepth ← ϕ
5:         for i in 1 to m do
6:             votesNdepth[i] ← GetVoteNDepth(i, ℓ)
7:         if IsPropSetConfirmed(votesNdepth) then    ▷ Refer Def. 4.5
8:             Π[ℓ] ← GetProposerSet(votesNdepth)     ▷ Refer Eq. 15
9:         else break
10:    // Ledger list decoding: Check if tx is confirmed in all ledgers
11:    prpBksSeqs ← Π[1] × Π[2] × ··· × Π[ℓ]          ▷ outer product
12:    for prpBks in prpBksSeqs do
13:        ledger = BuildLedger(prpBks)
14:        if tx is not confirmed in ledger then return False
15:    return True                                    ▷ Return true if tx is confirmed in all ledgers
16: // Return the vote of voter blocktree i at level ℓ and depth of the vote
17: procedure GetVoteNDepth(i, ℓ)
18:    voterMC ← vtTree[i].LongestChain()
19:    for vtrBk in voterMC do
20:        for prpBk in vtrBk.votes do
21:            if prpBk.level = ℓ then return (prpBk, vtrBk.depth)
22: procedure BuildLedger(propBlocks)                 ▷ Input: list of prop blocks
23:    ledger ← []                                    ▷ List of valid transactions
24:    for prpBk in propBlocks do
25:        refPrpBks ← prpBk.getReferredPrpBks()
26:        // Get all directly and indirectly referred transaction blocks.
27:        txBks ← GetOrderedTxBks(prpBk, refPrpBks)
28:        for txBk in txBks do
29:            txs ← txBk.getTxs()                    ▷ Txs are ordered in txBk
30:            for tx in txs do
31:                // Check for double spends and duplicate txs
32:                if tx is valid w.r.t to ledger then ledger.append(tx)
33:    return ledger
34: // Return ordered list of confirmed transactions
35: procedure GetOrderedConfirmedTxs()
36:    L ← ϕ                                          ▷ Ordered list of leader blocks
37:    for ℓ ← 1 to prpTree.maxLevel do
38:        votes ← ϕ                                  ▷ Stores votes from all m voter trees on level ℓ
39:        for i in ← 1 to m do
40:            votesNDepth[i] ← GetVotes(i, ℓ)
41:        if IsLeaderConfirmed(votesNDepth) then     ▷ Refer 4.1
42:            // Proposer block with maximum votes on level ℓ
43:            L[ℓ] ← GetLeader(votesNDepth)
44:        else break
45:    return BuildLedger(L)
```

B NOTATION

Let $H_i[r]$ and $Z_i[r]$ be the number of voter blocks mined by the honest and by the adversarial node in round r on the i-th voting tree respectively, where $i = 1, 2, .., m$. $H_i[r], Z_i[r]$ are Poisson random variables with means $(1-\beta)f_v\Delta$ and $\beta f_v\Delta$ respectively. Similarly, $H^p[r], Z^p[r]$ are the numbers of proposer blocks mined by the honest nodes and the adversarial node in round r respectively; they are also Poisson, with means $(1-\beta)f_p\Delta$ and $\beta f_p\Delta$ respectively. Finally, $H^t[r], Z^t[r]$ are the numbers of transaction blocks mined by the honest nodes and the adversarial node in round r respectively; they are also Poisson, with means $(1-\beta)f_t\Delta$ and $\beta f_t\Delta$ respectively. All the random variables are independent of each other.

C BITCOIN BACKBONE PROPERTIES

The authors in [9] define three important properties of the Bitcoin backbone: *common-prefix, chain-quality and chain-growth*. It was shown that, under a certain *typical execution* of the mining process, these properties hold, and the properties are then used to prove the persistence and liveness of the Bitcoin transaction ledger. These three properties, as well as the notion of a typical execution, were

global, and defined over the *entire* time horizon. While this is appropriate when averaging over time to achieve reliable confirmation, as for Bitcoin, it turns out that for the analysis of fast latency of Prism, where the averaging is over voter chains, we need to formulate finer-grained, *local* versions of these properties, localized at a particular round. Correspondingly, the event under which these local backbone properties are proved is also local, in contrast to the event of typical execution. We will focus on a single Bitcoin blocktree, with a mining rate of $\bar{f}$ per round, and we will use the model and notations introduced in Section 2. In addition, we will use the following notation from [9]: if C is a chain of blocks, then $C^{\lceil k}$ is the k-deep prefix of C, i.e. the chain of blocks of C with the last k blocks removed. Given two chains C and C', we say that $C \preceq C'$ if C is a *prefix* of chain C'.

Definition C.1 (Common-prefix property (CP)). *The k-deep common-prefix property is said to hold at round r if the k-deep prefix of the longest chain at round r remains a prefix of any longest chain in any future round.*

While the CP property in [9] is parameterized by a single parameter k, the property defined here is parameterized by two parameters k and r. It is a property that the prefix of the main chain at round r remains permanently in the main chain in the future.

Definition C.2 (Chain-quality property (CQ)). *The (μ, k)-chain-quality property is said to hold at round r if at most μ fraction of the last k consecutive blocks on the longest chain C at round r are mined by the adversary.*

The chain-quality property in [9] is parameterized by two parameters μ and k, however, the property defined here is parameterized by three parameters μ, k and r.

Definition C.3 (Chain-growth property (CG)). *Chain-growth property with parameters ϕ and s states that for any s rounds there are at least ϕs blocks added to the main chain during this time interval.*

We show that these three properties hold regardless of adversarial action, provided that certain events on the honest and adversarial mining processes hold. Let $r' := \frac{k}{2\bar{f}}$. Define the following events:

$$\mathsf{E}_1\left[r-r',r\right] := \bigcap_{a,b\geq 0} \Big\{Y[r-r'-a, r+b] - Z[r-r'-a, r+b] > \frac{(1-2\beta)k}{8}\Big\}$$

$$\mathsf{E}_2\left[r-r',r\right] := \left\{H\left[r-r',r\right] + Z\left[r-r',r\right] < k\right\}$$

$$\mathsf{E}_3\left[r-r',r\right] := \left\{X\left[r-r',r\right] > \frac{k}{6}\right\}$$

$$\mathsf{E}\left[r-r',r\right] := \mathsf{E}_1[r-r',r] \cap \mathsf{E}_2[r-r',r] \cap \mathsf{E}_3[r-r',r]. \quad (18)$$

As defined in Appen. B, $X\left[r-r',r\right]$ and $Y\left[r-r',r\right]$ are the number of successful and uniquely successful rounds respectively in the interval $[r-r',r]$, and $Z\left[r-r',r\right]$ is the number of blocks mined by adversary in the interval $[r-r',r]$. Therefore, the event $\mathsf{E}_1\left[r-r',r\right]$ implies that the number of uniquely successful rounds exceed the total blocks mined by the adversary by $\frac{(1-2\beta)k}{8}$ blocks for *all* the intervals containing the interval $[r-r',r]$. Event $\mathsf{E}_2\left[r-r',r\right]$ implies that the number successful rounds plus the total number of blocks mined by the adversary in the interval $[r-r',r]$ is less than k. Event $\mathsf{E}_3\left[r-r',r\right]$ implies that the number of successful rounds in the interval $[r-r',r]$ at least $\frac{k}{6}$.

Lemma C.4 (Lemma 6 [9]). *Suppose the k-th block, b, of a longest chain C was mined by a honest node in a uniquely successful round. Then the k-th block of a longest chain C', at possibly a different round, is either b or has been mined by the adversary.*

Lemma C.5 (Lemma 7 [9]). *Suppose that at round r_1 the longest chain is of length n. Then by round $r_2 \geq r_1$, the longest chain is of length of least $n + X[r_1, r_2]$.*

Lemma C.6. *Under the event $\mathsf{E}[r-r',r] \supseteq \mathsf{E}_2[r-r',r]$, the last k consecutive blocks of the longest chain C at round r are mined in at least r' consecutive rounds.*

The CG lemma below is the localized version of Thm. 13 from [9].

Lemma C.7 (Chain-growth). *Under event $\mathsf{E}[r-r',r] \supseteq \mathsf{E}_3[r-r',r]$, where $r' = \frac{k}{2\bar{f}}$, the longest chain grows by at least $\frac{k}{6}$ blocks in the interval $[r-r',r]$.*

We can slightly modify the proofs of Lem 14 and Thm 15 of [9] by localizing it to a particular round in order to prove our CP property.

Lemma C.8 (Common-prefix). *Under the event $\mathsf{E}[r_1 - r', r_1]$, where $r' = \frac{k}{2\bar{f}}$, k-deep comm.-prefix prop. holds at round r_1.*

Similarly we can modify the proof of Thm 16 of [9] by localizing it to a particular round in order to prove the chain-quality property.

Lemma C.9 (Chain-quality). *Under the event $\mathsf{E}[r-r',r]$, where $r' = \frac{k}{2\bar{f}}$, (μ, k)-chain quality prop. holds at round r for $\mu = \frac{7+2\beta}{8}$.*

We note that $\nu < 1$ if $\beta < 0.5$. Since the common-prefix, chain-quality and chain-growth properties are all proved assuming the event $\mathsf{E}[r-r',r]$ occurs, a natural question is how likely is its occurrence? The next lemma shows that the probability of it occurring exponentially approaches 1 as r' increases. This lemma will be heavily used in our analysis of security and fast confirmation.

Lemma C.10. *Let $\bar{f} \leq \frac{\log(2-2\beta)}{1-\beta}$.[8] For any r, $\mathbb{P}\left(\mathsf{E}^c\left[r-r',r\right]\right) \leq 4e^{-\gamma\bar{f}r'}$, where $r' = \frac{k}{2\bar{f}}$ and $\gamma = \frac{1}{36}(1-2\beta)^2$.*

Proof. From Lemmas C.11, C.12 and C.13, and Eqn. (18). □

Using Chernoff bound we obtain the following lemmas:

Lemma C.11. *For any r, $\mathbb{P}\left(\mathsf{E}_1^c\left[r-r',r\right]\right) \leq 2e^{-\frac{(1-2\beta)^2\bar{f}r'}{36}}$.*

Lemma C.12. *For any r, $\mathbb{P}\left(\mathsf{E}_2^c\left[r-r',r\right]\right) \leq e^{-\bar{f}r'}$.*

Lemma C.13. *For any r, $\mathbb{P}\left(\mathsf{E}_3^c\left[r-r',r\right]\right) \leq e^{-\frac{\bar{f}r'}{36}}$.*

D ORDERING: PROOFS OF THM. 4.1 AND 4.2

In Appendix C, we proved three chain properties – chain-growth, common-prefix and chain-quality – for the Bitcoin backbone under events defined in Equation (18). The voter blocktrees in Prism also follow the longest chain protocol, hence these three chain properties will directly hold for each of the m voter blocktree under the corresponding events:

[8] We will assume this constraint in all our results without stating it explicitly.

$$\mathsf{E}_{1,j}\left[r-r',r\right] := \bigcap_{a,b\geq 0}\Big\{Y_j[r-r'-a,r+b]- Z_j[r-r'-a,r+b] > \frac{(1-2\beta)k}{8}\Big\}$$

$$\mathsf{E}_{2,j}\left[r-r',r\right] := \left\{H_j\left[r-r',r\right] + Z_j\left[r-r',r\right] < k\right\}$$

$$\mathsf{E}_{3,j}\left[r-r',r\right] := \left\{X_j\left[r-r',r\right] > \frac{k}{6}\right\}$$

$$\mathsf{E}_j\left[r-r',r\right] := \mathsf{E}_{1,j}[r-r',r]\cap\mathsf{E}_{2,j}[r-r',r]\cap\mathsf{E}_{3,j}[r-r',r].$$

Note the similarity b/w above events and events in Eq. (18).

Typical event: For a given r', define the following event:

$$\mathsf{E}_j(r') := \bigcap_{\tilde{r}\geq r'}\ \bigcap_{0\leq r\leq r_{\max}} \mathsf{E}_j\left[r-\tilde{r},r\right]. \tag{19}$$

We note that by taking the intersection over various r, we have moved from a local version of the property to global version of the property, which will suffice for this section.

Lemma D.1. *For any j, $\mathbb{P}\left(\mathsf{E}_j^c(r')\right) \leq 4r_{\max}^2 e^{-\gamma\bar{f}_v r'}$, where $\gamma = \frac{1}{36}(1-2\beta)^2$.*

Proof. Use Lemma C.10 and apply union bound. □

Let the first proposer block at level ℓ appear in round R_ℓ. We will now prove common-prefix and chain-quality for the leader block sequence defined in Equation (11).

The **common prefix property** of the leader sequence gives us the *total ordering* in Prism . We derive this property using the common prefix and the chain-quality properties of the voter blocks.

Lemma D.2 (Common-prefix). *At round $r \geq R_\ell$, if every voter blocktree has a honest voter block mined after round R_ℓ which is at least k-deep, then w.p $1-\varepsilon_k$, the leader block sequence up to level ℓ is permanent i.e,*

$$\mathsf{LedSeq}_\ell(r) = \mathsf{LedSeq}_\ell(r_{\max}).$$

Here $\varepsilon_k \leq 4mr_{\max}^2 e^{-\gamma k/2}$ and $\gamma = \frac{1}{36}(1-2\beta)^2$.

Proof. From the typicality in Eqn. (19) and common-prefix Lemma C.8 we know for under the event $\mathsf{E}_j(r')$, for $k = 2r'\bar{f}_v$, the k-deep honest voter block and its ancestors permanently remain on the main chain of voter blocktree j. From Lemma D.1 we have $\mathbb{P}\left(\mathsf{E}_j^c(r')\right) \leq \frac{\varepsilon_k}{m}$. Thus, on applying union bound we conclude *all* the k-deep voter block on the m voter blocktrees are permanent w.p $1-\varepsilon_k$. Each of these honest voter blocks (along with their ancestors) have permanent votes on proposer blocks on all the levels $\ell' \leq \ell$ and this implies that the leader blocks up to level ℓ are permanent w.p $1-\varepsilon_k$. □

Therefore, to confirm leader blocks with $1-\varepsilon$ security, votes on all the m voter blocktrees should be at least $k = \frac{2}{\gamma}\log\frac{4mr_{\max}}{\varepsilon}$ deep. How long does it take to have (at least) a k-deep honest vote on *all* m voter blocktrees? The next lemma answers this question.

Lemma D.3. *By round $R_\ell + r_k$, wp $1-\varepsilon'_k$, all the voter blocktrees have an at least k-deep honest voter block mined after round R_ℓ, where $r_k \leq \frac{64k}{(1-2\beta)\bar{f}_v}$ and $\varepsilon'_k \leq 8mr_{\max}^2 e^{-\frac{\gamma\bar{f}_v r_k}{8}}$.*

Proof. Using the chain growth Lemma C.7 under the event $\mathsf{E}_j(r_k)$, we know that the main chain of voter blocktree j grows by $k_1 \geq \frac{r_k\bar{f}_v}{3}$ voter blocks. Next, using the chain-quality Lemma C.9 under the second event $\mathsf{E}_j\left(\frac{k_1}{2\bar{f}_v}\right)$, we know that at least $\frac{1-2\beta}{8}$ fraction of these k_1 voter blocks are mined by the honest users and the earliest of these voter block is at least k_2-deep, where $k_2 \geq \frac{(1-2\beta)k_1}{8} \geq \frac{(1-2\beta)\bar{f}_v r_k}{24} := k$. It is important to note that the depth k_2 is *observable* by the users. The probability of failure of either of these two events is

$$\leq\ \mathbb{P}\left(\mathsf{E}_j^c(r_k)\right) + \mathbb{P}\left(\mathsf{E}_j^c\left(\frac{r_k}{6}\right)\right) \overset{(c)}{\leq} \frac{\varepsilon'_k}{m}. \tag{20}$$

Inequality (c) is given by Lemma D.1 and on applying union bound on over m blocktree gives us the required result. □

Proof of Theorem 4.1. From Lemma D.3 we know that by round $R_\ell + r(\varepsilon)$, all the voter blocktrees will have a k-deep honest voter blocks wp at least $1-\frac{\varepsilon}{2}$ for $k \geq \frac{2}{\gamma}\log\frac{8mr_{\max}^2}{\varepsilon}$. Now applying Lemma D.2 for $k \geq \frac{2}{\gamma}\log\frac{8mr_{\max}^2}{\varepsilon}$, we obtain that all these honest voter blocks are permanent w.p $1-\frac{\varepsilon}{2}$. On combining these two, we obtain that by round $R_\ell + r(\varepsilon)$ the leader block sequence up to level ℓ is permanent w.p $1-\varepsilon$.

Average Case: The conf. policy in Lem. D.3 is under worst case adversarial attack: when there are two (or more) proposer blocks at a given level have equal number of votes. Consider an 'typical case' with two proposer blocks with $2m/3$ and $m/3$ votes. One can see that we don't need to guarantee permanence of all the m votes but a weaker guarantee suffices: permanence of $m/2$ of the $2m/3$ votes. This weaker guarantee can be achieved within a few rounds.

Corollary D.4. Bitcoin*'s latency is the time required to mine a single honest $\frac{1}{\varepsilon}$-deep block on a voter chain of* Prism *and it requires lesser than $\frac{2304}{\bar{f}_v(1-2\beta)^2}\log\frac{8r_{\max}^2}{\varepsilon}$ rounds to provide $1-\varepsilon$ reliability to confirm blocks and the transactions in it.*

We have proved consistency via leader common prefix property. To show liveness, we now show that honest proposer blocks become leaders with good enough frequency.

Definition D.5 (Leader-sequence-quality). *The (μ,k)-leader-sequence-quality property holds at round r if at most μ fraction of the last k consecutive leader blocks on the proposer blocktree at round r are mined by the adversary.*

Lemma D.6 (Leader-sequence-quality). *The (μ,k)-leader-sequence-quality property holds at round r for $\mu = \frac{7+2\beta}{8}$ w.p at least $1-4r_{\max}^2 e^{-(1-2\beta)^2 k/72}$.*

Proof. Since leader block sequence is not a chain, we have to slightly modify the proof of chain quality from [9]. The honest users have higher mining rate compared to the adversary and this is used to show that the proposer blocks mined by the honest users will frequently enter the leader sequence. □

The leader sequence quality defined in D.5 is parameterized by two parameters r and k, whereas its counterpart definition of chain quality in [9], is parameterized only by a single parameter k. Even

though our definition of 'quality' is a weaker, we show that it is suffices to ensure liveness.

Proof of Theorem 4.2. Let $k := \frac{2048}{(1-2\beta)^3}\log(\frac{32mr_{\max}}{\varepsilon})$ and $k_1 := \frac{8k}{1-2\beta}$. Using Lemma D.6 we know that w.p at least $1-\varepsilon/4$, the last k_1 leader blocks have at least k honest leader blocks. From Lemma C.6 and D.1, w.p at least $1-\varepsilon/4$, the deepest of these k honest leader block was proposed before the round $r - \frac{k}{2\bar{f}_v}$. Now using Theorem 4.1, this deepest honest leader block is permanent w.p $1-\varepsilon/4$. Therefore, the honest transaction will be permanently added to the blockchain after k_1 proposer blocks are mined. Using chain growth Lemma C.7, we know that the k_1 proposer blocks will be mined in no more than $\frac{3k_1}{\bar{f}_v}$ rounds w.p $1-\varepsilon/4$. Therefore, w.p $1-\varepsilon$, the tx will be part of the permanent leader sequence in

$$\frac{3k_1}{\bar{f}_v} = \frac{3\times 2^{14}}{(1-2\beta)^3\bar{f}_v}\log\frac{32mr_{\max}}{\varepsilon} \text{ rounds.} \tag{21}$$

The constants in Eqn (21) have not be optimized for simplicity. The scaling w.r.t $1-2\beta$, $\bar{f}_v$ and $\log\frac{1}{\varepsilon}$ is the main take away.

E CONFIRMATION: PROOF OF THM. 4.6

E.1 Voter chain properties

In Appendix D, we proved the common-prefix and the leader-sequence-quality properties by requiring the typical event defined in Equation (19) to hold for *every* voting chain, i.e. at the *microscopic* scale. The typicality of each such event was obtained by averaging over rounds and as a consequence the confirmation of leader blocks with $1-\varepsilon$ guarantee required averaging over $O(\log\frac{1}{\varepsilon})$ rounds. In this section we obtain faster confirmation time by relaxing the notion of typicality to a notion of *macroscopic* typicality, one which concerns the mining processes of a large fraction of the voter chains. This event guarantees macroscopic versions of the chain-growth, common-prefix and chain-quality properties. That is, these properties are guaranteed to be satisfied by a large fraction of the voter chains, but *not* all. These macroscopic properties of the voting chains turn out to be sufficient to allow fast confirmation. For this whole section, we will define:

$$\gamma := \frac{1}{36}(1-2\beta)^2,\ c_1 := \frac{1-2\beta}{16},\ r_{\min} := \frac{2\log\frac{200}{\gamma c_1}}{\gamma\bar{f}_v}$$
$$k_{\min} := \frac{4}{\gamma}\log\frac{200}{\gamma c_1},\ \rho_{r'} := \max\left(\frac{c_1}{1+4\bar{f}_v r'}, \frac{(1-2\beta)c_1}{1+32\log m}\right)$$
$$\delta_k := \max\left(\frac{c_1}{1+2k}, \frac{(1-2\beta)c_1}{1+32\log m}\right),\ \varepsilon_m := r_{\max}^2 e^{-\frac{(1-2\beta)c_1 m}{2+64\log m}}. \tag{22}$$

Lemma E.1 (Macroscopic Typicality). *The macroscopic typical event* T *defined below occurs with probability* $1-\varepsilon_m$.

$$\mathsf{T}[r-r',r] := \left\{\frac{1}{m}\sum_{j=1}^{m}\mathbf{1}\big(\mathsf{E}_j[r-r',r]\big) \geq 1-\delta_k\right\}$$
$$\mathsf{T} := \bigcap_{0\leq r\leq r_{\max}, r'\geq r_{\min}} \mathsf{T}[r-r',r], \quad \text{where } r' = \frac{k}{2\bar{f}_v}$$

.

Proof. For a fixed r, r', the indicator random variables $\mathbf{1}\big(\mathsf{E}_j^c[r-r',r]\big)$ are identical and independent $\forall j \in [m]$. Using Chernoff bounds one can obtain the required result. □

Lemma E.2 (Macroscopic Chain-growth). *Under the event* T, *for* $k \geq k_{\min}$, $r' = \frac{k}{2\bar{f}_v}$, *the longest chain grows by at least* $\frac{k}{6}$ *blocks in the interval* $[r-r', r]$ *on at least* $1-\delta_k$ *fraction of voter blocktrees.*

Proof. From the typicality Lemma E.1, we know that under the event T, $\frac{1}{m}\sum_{j=1}^{m}\mathbf{1}\big(\mathsf{E}_j[r-r',r]\big) \geq 1-\delta_k$. Applying Lem. C.7 on events $\mathsf{E}_j[r-r',r]$, $\forall j\in[m]$ gives us the result. □

Lemma E.3 (Macroscopic Common-prefix). *Under event* T, *for* $k \geq k_{\min}$ *and* $r' = \frac{k}{2\bar{f}_v}$, *the* k*-deep common-prefix prop. holds at round* r *for at least* $1-\delta_k$ *fraction of voter blocktrees.*

Proof. Similar to the proof in E.2 □

Lemma E.4 (Macroscopic Chain-quality). *Under the event* T, *for* $k \geq k_{\min}$ *and* $r' = \frac{k}{2\bar{f}_v}$, *the* (μ,k)*-chain quality property holds at round* r *for* $\mu = \frac{7+2\beta}{8}$ *for at least* $1-\delta_k$ *fraction of voter blocktrees.*

Proof. Similar to the proof in E.2 □

In Appendix D, we used microscopic properties for *long interval* of rounds for each voter chain to obtain the common-prefix and the leader sequence quality properties. Here we change our strategy: we use macroscopic properties of the voter chains to obtain the common-prefix and the leader sequence quality properties for *short interval* of rounds and this directly translates to short latency.

E.2 Fast list confirmation policy

Consider the definitions from Section 4.5.1. Note that $V_n^k(r)$ and $V_{-n}^k(r)$ are observable quantities. The following lemma bounds the future number of votes on a proposer block.

Lemma E.5. *With probability at least* $1-\varepsilon_m$, *the number of votes on any proposer block* p_n *in any future round* $r_f \geq r$, $V_n(r_f)$, *satisfies*

$$\underline{V}_n(r) \leq V_n(r_f) \leq \overline{V}_n(r),$$

Proof. From typicality in Lemma E.1 we have $\underline{V}_{-n}(r) := \max_{k\geq k_{\min}}(V_{-n}^d(r) - \delta_k m)_+$. At most $(V_{-n}(r) - \underline{V}_{-n}(r))$ votes can be removed from proposer blocks in the set $\mathcal{P}_\ell(r) - \{p_n\}$ and added to the proposer block p_n. Also the $U_\ell(r)$ voter blocktrees which have not yet voted could also vote on block p_n. Combining these both upper bounds on $V_n(r_f)$. □

Any private block $p_{\text{private}} \notin \mathcal{P}_\ell(r)$ by definition has zero votes at round r. The future number of votes on the proposer block p_{private} w.p $1-\varepsilon_m$ satisfies

$$V_{\text{private}}(r_f) \leq \overline{V}_{\text{private}}(r) := m - \sum_{p_n\in\mathcal{P}_\ell(r)}\underline{V}_n(r) \qquad \forall r_f \geq r,$$

Refer 4.5 for the fast list confirmation policy. Fig. 9 illustrate one such example. The definition of $\Pi_\ell(r)$ is *precisely designed* to prevent private proposer blocks from becoming the leader blocks in the future rounds.

Lemma E.6. *If the proposer sets are confirmed for all levels $\ell' \le \ell$ by round r, then w.p $1-\varepsilon_m$, by contradiction, the final leader sequence up to level ℓ satisfies* $p^*_{\ell'}(r_{\max}) \in \Pi_{\ell'}(r) \quad \forall \ell' \le \ell$.

Lemma E.6 proves that the proposer sets obtained via the fast list confirmation policy contains the final leader blocks. The natural question is: how long does it take to satisfy the constraint for the fast list confirmation? We answer this next.

E.3 Latency

We use liveness and consistency of the voter chains to show that the proposer blocks at level (say) ℓ accumulates enough votes in constant number of rounds to satisfy the confirmation policy by triggering Eqn. (14). The first proposer block at level ℓ appears in round R_ℓ. First, Lem. E.7 characterizes the rate at which votes are accumulated for prop. blocks at level ℓ. Second, Lem. E.9 uses this result to prove that the confirmation policy for level ℓ is satisfied in constant number of round after R_ℓ. Lastly, Lem. E.10 uses this to show that the conf. policy is satisfied for all levels upto ℓ in constant number of rounds after R_ℓ and then we invoke Lemma E.6 to prove common-prefix type property. For simplicity we assume that the proposer blocktree mining rate $\bar{f}_p = \bar{f}_v$. Define $\Delta_0 := \frac{12 r_{\min}}{1-2\beta}$.

Lemma E.7. *By round $r = R_\ell + \Delta_r$, for $\Delta_r \ge \Delta_0$, w.p $1-\varepsilon_m$, at least $1 - 4\rho_{\Delta_r}$ fraction of the voter blocktrees have an honest voter block which is mined after round R_ℓ and is at least k-deep on the main chain. Here $k \ge \max(\frac{(1-2\beta)\bar{f}_v \Delta_r}{24}, k_{\min})$.*

Proof. We use chain growth lemma E.2, along with chain quality lemma E.4 of the voter chains to prove this. □

Define $N_\ell(r) := |P_\ell(r)|$ as the number of proposer blocks on level ℓ at round r and let $c_1 = \frac{1-2\beta}{16}$ and $c_2 := \frac{16}{\bar{f}_v(1-2\beta)^3}$.

Lemma E.8. *The proposer set at level ℓ can be confirmed w.p $1-\varepsilon_m$ in round $r = R_\ell + \Delta_r$ for for $\Delta_r \ge \Delta_0$ if*

$$\textit{Case 1}: \quad N_\ell(R_\ell + \Delta_r) + 1 < \frac{c_1}{\rho_{\Delta_r}}, \ \textit{or} \quad \textit{Case 2}: \quad \Delta_r = c_2 m.$$

Proof. For Case 1, using Lemma E.7 round $r = R_\ell + \Delta_r$, at least $1 - 4\rho_{\Delta_r}$ fraction of voter blocktrees have k-deep votes on proposer blocks in $\mathcal{P}_\ell(r)$ where $k \ge \frac{(1-2\beta)\bar{f}_v\Delta_r}{24}$. Using Lemma E.5, we have $\sum_{p_n \in \mathcal{P}_\ell(r)} \underline{V}_n(r) \ge m(1 - 4\rho_{\Delta_r} - \delta_k)$, where the constant δ_k satisfies $\delta_k \le \frac{12\rho_{\Delta_r}}{(1-2\beta)}$. Without loss of generality we have

$$\underline{V}_1(r) \ge \frac{m}{N_\ell(r)}\left(1 - \frac{16\rho_{\Delta_r}}{1-2\beta}\right) \tag{23}$$

$$\overline{V}_{\text{private}}(r) < m - \sum_{p_n \in \mathcal{P}_\ell(r)} \underline{V}_n(r) < \frac{(1-2\beta)\rho_{\Delta_r}}{16}. \tag{24}$$

From Equations (23) and (24), it is easy to see that

$$N_\ell(r) + 1 < \frac{16}{(1-2\beta)\rho_{\Delta_r}} \Longrightarrow \underline{V}_1(r) > \overline{V}_{\text{private}}(r),$$

and therefore, the proposer set at level ℓ can be confirmed by round r. Case 2 is a corollary of Theorem 4.1 □

Let us define the random variable:

$$R^{\text{stop}}_\ell := \min \Delta_r > \Delta_0 \ s.t \ N_\ell(R_\ell + \Delta_r) + 1 < \frac{c_1}{\rho_{\Delta_r}}. \tag{25}$$

$R^{\text{stop}}_\ell = \infty$ if the above inequality is not satisfied for any Δ_r.

Lemma E.9. *The proposer set at level ℓ can be confirmed by round $R_\ell + \min(R^{stop}_\ell, c_2 m)$ and we have*

$$\mathbb{E}[\min(R^{stop}_\ell, c_2 m)] \le \frac{13}{(1-2\beta)} r_{\min} + \frac{48}{\bar{f}_v(1-2\beta)^3 m^3}.$$

Proof. In expectation $N_\ell(r)$ grows slower than $\frac{c_1}{\rho_{\Delta_r}}$ and we use this to show that R^{stop}_ℓ is $\frac{12}{(1-2\beta)} r_{\min}$ in expectation conditioned on $R^{\text{stop}}_\ell < \Delta_{\max}$, where $\Delta_{\max} = \frac{8 \log m}{\bar{f}_v(1-2\beta)}$. We then show that $\mathbb{P}(R^{\text{stop}}_\ell > \Delta_{\max}) = e^{-(1-2\beta)f_v\Delta_{\max}/2}$. □

Lemma E.9 upper bounds the expected number of rounds to confirm the proposer block set at level ℓ. However, our goal in Lemma E.6 is to confirm proposer set for *all* the levels $\ell' \le \ell$. All the proposer set up to level ℓ are confirmed in the following number of rounds:

$$R^{\text{conf}}_\ell := \max_{\ell' \le \ell} \left(R_{\ell'} + \min(R^{\text{stop}}_{\ell'}, c_2 m) - R_\ell\right) \tag{26}$$

Expression (26) is a maximum of random variables associated with each level up to level ℓ. It turns out that the max term is dominated by random variable associated with level ℓ and we have

Lemma E.10. *All the prop. sets up to level ℓ are confirmed in the following number of rounds in expectation*

$$\mathbb{E}[R^{conf}_\ell] \le \frac{13}{(1-2\beta)} r_{\min} + \frac{256}{(1-2\beta)^6 \bar{f}_v m^2}$$

Proof of theorem 4.6: Substitute $r_{\min}$ from (22) in above lemma.

F PROOF OF THEOREM 4.7

Let the transaction tx enters the system in round r and let ℓ be the last level on the proposer blocktree which has proposer blocks at round r. Define $\ell^* := \max\left(\tilde{\ell} \le \ell \ \ s.t \ \ \text{the honest users mined the first proposer block on level } \tilde{\ell}\right)$. Let r^* be the round in which the first proposer block was mined on level ℓ^*. From the definition of ℓ^* we have the following two observations a) All the proposer blocks on levels greater or equal to ℓ^* are mined on or after round r^* by definition, b) the adversary has mined at least one proposer block on all levels in $[\ell^*, \ell]$. Define $\Delta_0 := \frac{12 r_{\min}}{1-2\beta}$. For $r_f \ge r$, let us define the following event: $A_{r_f} = \left\{Y^p[r^*, r_f - \Delta_0] - Z^p[r^*, r_f] > 0\right\}$.

Lemma F.1. *If event A_{r_f} occurs, then the transactions tx is included in a block b which is proposed in round $r(b) \le r_f - \Delta_0$ and confirmed as a leader block by round r_f.*

Proof. From the observations in the earlier paragraph. □

Lemma F.2. *Let $R_f := \min r_f \ge r \ \ s.t \ \ A_{r_f}$ occurs. Then,*

$$\mathbb{E}[R_f - r] \le \frac{24(1-\beta) r_{\min}}{(1-2\beta)^2} \le \frac{2592}{(1-2\beta)^3 \bar{f}_v} \log \frac{50}{(1-2\beta)}.$$

Proof. $Y^p[r^*, r_f - \Delta_0] - Z^p[r^*, r_f]$ as a random walk and $\mathbb{E}[R_f - r]$ is the expected time for this random walk to hit zero. □

From Lemma F.1 and F.2, we conclude that a transaction, which is part of a transaction block mined in round r, is referred by a proposer block at level (say) ℓ and the leader block at this level confirmed before round $r + \frac{2592}{(1-2\beta)^3 \bar{f}_v} \log \frac{50}{(1-2\beta)}$ in expectation. This proves the main claim of Theorem 4.7.