Permissionless Verifiable Information Dispersal (Data Availability for Bitcoin Rollups)

Ben Fisch[1] Arthur Lazzaretti[1] Zeyu Liu[1] Lei Yang[2*]

[1]Yale University
[2]MIT CSAIL

Abstract

Rollups are special applications on distributed state machines (aka blockchains) for which the underlying state machine only logs, but does not execute transactions. Rollups have become a popular way to scale applications on Ethereum and there is now growing interest in running rollups on Bitcoin. Rollups scale throughput and reduce transaction costs by using auxiliary machines that have higher throughput and lower cost of executing transactions than the underlying blockchain. State updates are periodically posted to the underlying blockchain and either verified directly through succinct cryptographic proofs (zk rollups) or can be challenged for a defined period of time in a verifiable way by third parties (optimistic rollups).

However, once computation is removed as a bottleneck, communication quickly becomes the new bottleneck. The critical service the underlying blockchain provides in addition to verification is *data availability*: that necessary data can always be recovered upon request. While broadcasting transaction data is one way to ensure this, it requires communication blowup linear in the number of participating nodes. Verifiable information dispersal (VID) systems achieve sublinear blowup in the same participation model and the same security assumptions as Ethereum, where all nodes have a strong public-key identity. It was not known how to do so in the same *permissionless* model as Bitcoin, where participants are unauthenticated and participation is dynamic.

We construct a VID system that is secure under the same model as Bitcoin, with one minimal additional requirement on the existence of reliable participants. Our system uses a state machine replication (SMR) protocol (e.g., Bitcoin) as a black box, and is therefore backward compatible. We implemented the system on top of Bitcoin core with the Regression Test Network (regtest), and our analysis shows that it reduces communication costs by more than $1,000\times$ and latency by more than $10\times$.

1 Introduction

Rollups are special applications on distributed state machines (aka blockchains) for which the underlying state machine only logs, but does not execute transactions [26, 33]. They are said to operate on the Layer 2 (L2) on top of the underlying Layer 1 (L1) blockchain. Rollups may encompass a single application, or may themselves be a general purpose virtual machine (VM) that hosts other applications or even other rollups. Recently, rollups have become a popular way to scale applications on Ethereum [35] and there is now growing interest in running rollups on Bitcoin [27].

Rollups introduce one or more auxiliary servers that execute transactions and periodically post underlying state updates to the L1 along with compressed transaction data. There are two flavors for handling these updates. *Validity rollups* (aka zk rollups [34, 14]) create *succinct cryptographic proofs* that are immediately verified by the L1 upon receipt of the update, proving that the update is consistent with the correct execution of these transactions. In *optimistic rollup* [24] the L1 accepts updates optimistically, but for a defined window

[*]Authors are ordered alphabetically.

of time will accept challenges from third parties who may claim the update was incorrect. This initiates an interactive game between the party that initially posted the update and the challenger (called a fault proof) in which the L1 is able to efficiently judge the integrity of the challenge. The update may be rolled back if determined to be incorrect.

Rollups scale throughput and reduce transaction costs by using auxiliary machines that have higher throughput and lower cost of executing transactions than the underlying blockchain [9]. Blockchains like Ethereum or Bitcoin have relatively low throughput and high cost because (a) the hardware requirements for operating a node are deliberately capped to encourage large and diverse participation (decentralization) and (b) the price a user pays for a transaction must cover the operational cost of *all* participating nodes executing that transaction, which is redundant.[1]

Once computation (executing transactions) is removed as a bottleneck, communication (broadcasting transaction data to all L1 nodes) quickly becomes the new bottleneck [29]. Indeed, posting transaction data to Ethereum is the primary cost for rollups today contributing to transaction prices. However, since the L1 nodes do not need to execute rollup transactions, it is actually unnecessary for every single node to receive a full copy of the transaction data. Rather, the L1 nodes only need to collectively ensure *data availability*: that at any point in time, any client can request the system to reconstruct the correct state. In optimistic rollups, the available data must also include sufficient information to initiate a fraud challenge if necessary (for example, the transaction list) [9]. In a system like Ethereum, erasure codes make it easy to achieve data availability with significantly less communication than broadcasting the data to all nodes, assuming a maximum threshold of corrupt nodes [10]. Specifically, with N nodes where at most an f fraction of these nodes are corrupt, and an optimal erasure code with rate $1 - f$, it suffices for each node to receive a $1/N$ slice of the encoded data since there exist at least $N(1-f)$ non-corrupted nodes that are able to reconstruct the original data. The total required communication in this case for a single β-bit blob of data is only $O(\beta/(1-f))$ rather than $O(\beta \cdot N)$. Ethereum is undergoing changes to its core protocol to enable this [2] and alternative data availability systems such as EigenDA [3] and Celestia [1] are already in use by rollups.

In so-called *proof-of-stake* systems like Ethereum, data availability with sublinear[2] communication can be achieved under the same security assumptions as the underlying consensus protocol because both rely on the fact that the set of participants are identified by public-keys, that there is a known upper bound on the number of corrupted participants, and that non-corrupt nodes stay online for sufficiently long periods of time.

Bitcoin's consensus protocol, which leverages *proof-of-work*, has a unique set of characteristics that make it quite different from Ethereum's. Bitcoin (1) is dynamically available under the dynamic participation of nodes; and (2) does not require nodes to authenticate themselves in order to participate (i.e., is permissionless). In particular, unlike Ethereum, participating nodes do not need to preregister their keys.[3] For Bitcoin rollups to enjoy similar cost savings as Ethereum rollups (i.e., reduce both redundant computation and communication) without compromising on these key characteristics, they require a protocol for data availability that achieves sublinear communication under these same set of assumptions. Prior to our present work, *no* data availability system with sublinear communication has been shown to achieve this.

In this paper, we close the gap by introducing the first data availability protocol that tolerates dynamic participation and requires no authentication. The only additional assumption we need is the existence of reliable participants (i.e., there are nodes that come back online even if going offline, within a non-deterministic period of time). We believe this additional condition is the minimum requirement (to allow reducing the communication cost), as discussed in Section 4.2.

As in existing solutions, participants in our protocol download only small slices of the proposed block,

[1]Today high transaction costs on Ethereum seem to be due to congestion (low throughput relative to demand) rather than the dollar operational costs of Ethereum nodes collectively. This may also be attributed to Ethereum rewards (inflation) subsidizing operational costs.

[2]Sublinear in the data size per participating node.

[3]In Ethereum, a node joins the system by purchasing ETH and then "staking" at least 32 ETH [20], which technically requires the cooperation of the counterparty selling ETH and the consensus itself to approve the participant as a member of the staking set. On the other hand, if Ethereum's consensus is censorship-free then it effectively functions as a permissionless protocol as well.

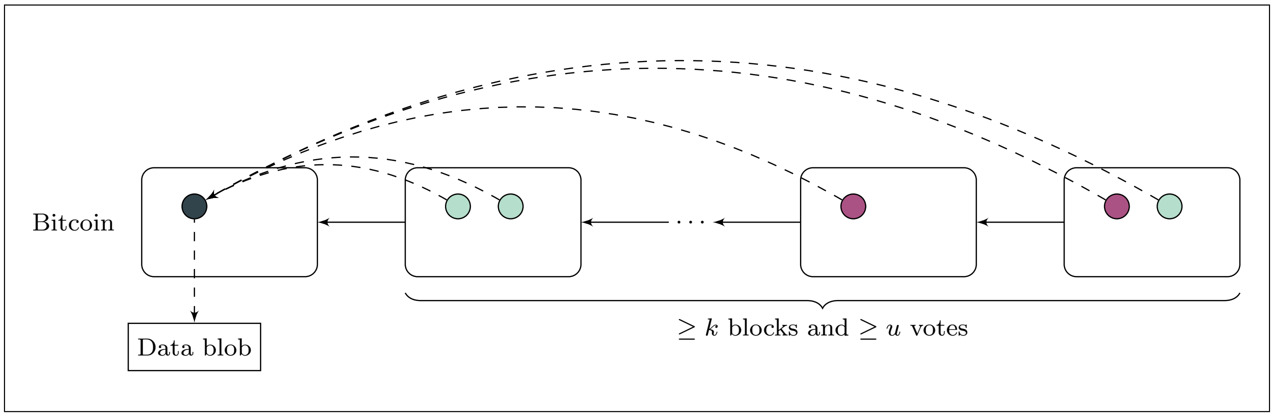


Figure 1: Our data availability protocol bootstrapped on an existing SMR protocol (rectangles and arrows). Circles are messages in our protocol, which are submitted to the SMR protocol as transactions. The black circle is the commitment of a data blob. The green circles are "yes" votes. The purple circles are "no" votes.

and vote to report whether they succeed. The protocol confirms that the block has been durably stored after seeing a sufficiently large fraction of "yes" votes, which implies that a sufficient number of slices are stored by honest participants. However, unlike existing protocols that use predefined membership sets or proof of stake to decide who has the power (and the responsibility) to vote, our voting process is secured with proof of work, which naturally preserves the permissionlessness of Bitcoin.

In our construction, a key challenge is to ensure all validators agree on whether a block is durably stored or not, allowing them to collectively skip unavailable blocks (which would halt the rollup) while staying in agreement on the rollup history. This implies solving the (binary) byzantine agreement (BA) problem, which does not have an efficient solution in the permissionless setting either[4]. Our strategy is to use the state machine replication (SMR) protocol of the layer-1 blockchain to bootstrap BA, which is possible since SMR and BA are equivalent problems. Doing so has two benefits: first, the L1 blockchain already exists, so we do not need to instantiate a new protocol or modify L1; and second, by reusing the L1 we do not bring extra security assumptions. When using Bitcoin as the L1, the resulting BA protocol retains its permissionless property and almost retains dynamic participation.

1.1 Technical Overview of Our Result

We now give an overview of our construction, also illustrated in Figure 1. In the protocol, participants post various messages to an underlying SMR protocol by submitting transactions that encode the messages. We assume the SMR protocol is safe, i.e., all participants agree on the content and the ordering of the posted messages (as otherwise, it is pointless to work over such an SMR protocol).

To propose a data blob (rollup block), the proposer posts a commitment to the data blob to the SMR protocol. This is represented as the black circle in Figure 1.

Once this commitment is included in the SMR protocol, other participants will see the commitment and start querying the proposer for data samples of the proposed blob. Upon getting a response, a participant verifies if the reply is consistent with its request and the commitment posted to the SMR protocol. If so, it posts a "yes" vote to the SMR protocol, represented as a green circle in Figure 1. Otherwise, it submits a "no" vote, represented as a purple circle in Figure 1.

In order to vote, participants must also include a small proof of work in their votes[5]. This is an anti-Sybil mechanism that ensures that malicious participants cannot produce too many fake votes, and that the

[4]See Remark A.1 for discussion about using existing permissionless BA.

[5]Note that the proof of work can be integrated with the underlying SMR protocol if the underlying one is also proof-of-work-based. See Remark 5.1 for a more detailed discussion.

number of votes each party casts is proportional to its computational power.

Participants repeat the sampling and voting procedures, until they see both of the following termination conditions are met: (1) there are at least k new blocks proposed after the commitment was posted in the underlying SMR protocol, and (2) there are more than u votes (either "yes" or "no") cast on the commitment. This ensures that enough votes are cast to accurately reflect the sampling results of the participants, and there are enough blocks to include the honestly generated votes. When both conditions are satisfied, participants decide whether a block is available by examining if the difference between the number of "yes" and "no" votes is larger than a given threshold.

Specifically, we show that with carefully analyzed threshold and k, u, the following properties hold: (i) an honestly proposed blob is always confirmed as available; (ii) every honest party will always agree on whether or not a blob is available; and (iii) every honest party will eventually be able to reconstruct the proposed blob if they decide it is available. In Section 5.2, we show our construction in more detail and how to extend the idea to process multiple data blobs simultaneously, instead of voting for a single blob every (approximately) k-block interval.

The resulting scheme is both asymptotically and concretely much more efficient than posting the blob directly to layer-1, as evidenced by our analysis and implementation. Specifically, when instantiating the underlying SMR with Bitcoin, the asymptotic communication cost (per participant) grows sublinear in the data blob size while directly posting the blob using Bitcoin takes linear size. Concretely, the communication cost is reduced by more than 1000x and the latency is also reduced by more than 10x (detailed evaluation in Section 6).

2 Related Works

Data availability protocols vary in the interfaces and the properties they provide, but in general, they define at least three procedures: `disperse`, where a proposer uploads a new block; `verify`, where a validator checks if a block has been successfully dispersed; and `retrieve`, where a validator downloads a block that has been successfully dispersed. We defer the formal definition of them and the properties they must satisfy in our protocol to Section 4.1. But at the minimum, a secure data availability protocol should ensure that validators can always `retrieve` blocks that are successfully `dispersed` as indicated by the `verify` primitive. This is important because the output of `verify` usually decides whether a block should be included in the final ledger of the blockchain [6, 37]. If an unretrievable block is included, the blockchain will lose liveness.

On a high level, there are two classes of data availability protocols, and they differ in whether nodes are guaranteed agreement on the result of `verify`. In the first class of protocols, different honest nodes may get different results when calling `verify` for the same block. The most prominent example of such protocols is Data Availability Sampling (DAS) [7], where the calling node requests a random subset of the block's chunks. If all the requests are fulfilled, the node deems the block retrievable, and vice versa. Apparently, when the node requests enough chunks, it can guarantee with arbitrarily low error probability that if all the requested chunks are fulfilled, at least a constant fraction (e.g., 1/4) of the block's chunks are retrievable, which is sufficient to recover the entire block when the block is encoded with a maximum distance separable erasure code of rate 1/4. However, when more than 1/4 but not all of the chunks are retrievable (e.g., when the proposer is adversarial), nodes may get different results depending on the random subset they request. In this case, their Verify outputs different results. This is problematic because nodes will disagree on whether the block should be included in the ledger. To fix this issue, DAS proposes to let nodes collaboratively recover the unretrievable chunks *after* their Verify process. While this is theoretically possible, the requirement of recovering causes huge overheads: to the best of our knowledge, there has been no practical instantiation of this idea. Thus, we need to avoid having to recover the whole block in order to agree.

The second class of protocols guarantees agreement on the result of `verify`. Instead of letting each node `verify` locally, these protocols use a permissioned set of servers with an honest super majority to collectively decide if a block is retrievable. Each server is assigned to store a subset of chunks, and it signs a vote after it fulfills the assignment. Nodes invoking `verify` simply checks if there are enough signed votes; if so, the block is deemed retrievable, because enough chunks are stored by honest servers. As long as every node sees every vote (which is easy to achieve by reliable-broadcasting the votes), they will agree on the result. Examples of this class are verifiable information dispersal (VID) protocols [11, 22, 36, 28]. However, as mentioned earlier, using a predefined set of servers means the protocol is *not* permissionless. Also, the protocol loses liveness if too many servers leave the system before fulfilling their responsibility to store chunks and vote, meaning the protocol is *not* dynamically available either.

3 Background

3.1 Data Availability Sampling

One important primitive we use in our construction is a Data Availability Sampling (DAS) protocol introduced in [7] and formally abstracted as a primitive in [21].[6] At a high-level, this allows a set of parties to run the following procedure. First, one party proposes and commits to a data blob $\vec{b}$ by outputting a commitment to $\vec{b}$ denoted com. Then, other parties can request random, small samples of this blob $\vec{b}$. The proposer replies to these samples with a reply R. The parties that requested samples can verify that R is indeed a correct random sample with respect to com and the randomness it picked, therefore the committer can not cheat by replying with an incorrect answer. Additionally, enough samples allow the parties to recover the entirety of $\vec{b}$. We recall the formal definition as follows (which is a simplified variant of the definition introduced in [21], where we fix the alphabet to be bit strings and combine soundness with consistency). Data availability has five PPT algorithms (where Commit, $\mathsf{Reconstruct}$, and Vrfy are deterministic).

- $\mathsf{pp} \leftarrow \mathsf{Setup}(\lambda, \epsilon)$: takes a security parameter λ, a recovery parameter $\epsilon > 0$; outputs a public parameter pp.
- $\mathsf{com} \leftarrow \mathsf{Commit}(\mathsf{pp}, \vec{b})$: takes a public parameter pp and a data blob $\vec{b} \in \{0,1\}^*$; outputs a commitment com.
- $R \leftarrow \mathsf{Sampling}(\mathsf{pp}, \vec{b}; r)$: takes a public parameter pp and a data blob $\vec{b} \in \{0,1\}^*$ together with some randomness source r; outputs a sample R.
- $b \leftarrow \mathsf{Vrfy}(\mathsf{pp}, \mathsf{com}, R)$: takes a public parameter pp, a commitment com, and a sample R; outputs a bit $b \in \{0,1\}$.
- $\vec{b}' \leftarrow \mathsf{Reconstruct}(\mathsf{pp}, \{R_i\}_{i\in[\epsilon]})$: takes a public parameter pp, ϵ samples; outputs data blob $\vec{b}'$.

With the following properties.

- (Correctness) For any $\lambda > 0, \epsilon > 0$, let $\mathsf{pp} \leftarrow \mathsf{Setup}(\lambda, \epsilon)$, let $r_1, \ldots, r_\epsilon$ be ϵ honestly produced randomness source; for any $\vec{b} \in \{0,1\}^*$ let $R_i \leftarrow \mathsf{Sampling}(\mathsf{pp}, \vec{b}; r_i)$, let $\mathsf{com} \leftarrow \mathsf{Commit}(\mathsf{pp}, \vec{b})$, it holds that $\Pr[\mathsf{Vrfy}(\mathsf{pp}, \mathsf{com}, R_i) = 1] = 1 - \mathsf{negl}(\lambda)$ for all $i \in [\epsilon]$.
- (Security) For any $\lambda > 0, \epsilon > 0, \vec{b} \in \{0,1\}^*$, let $\mathsf{pp} \leftarrow \mathsf{Setup}(\lambda, \epsilon)$, $\mathsf{com} \leftarrow \mathsf{Commit}(\mathsf{pp}, \vec{b})$; for any PPT adversary $\mathcal{A}$, let $(\mathsf{com}, (R_{1,i})_{i\in N_1}, (R_{2,i})_{i\in N_2}) \leftarrow \mathcal{A}(\mathsf{pp}, \vec{b})$, it holds that for any $S_1 \in [N_1], S_2 \in [N_2]$, where $|S_1| = |S_2| = \epsilon$, if for all $j \in [1,2]$, $\mathsf{Vrfy}(\mathsf{pp}, \mathsf{com}, (R_{j,i})_{i\in S}) = 1$, let $\vec{b}_j \leftarrow \mathsf{Reconstruct}(\mathsf{pp}, \{R_{j,i}\}_{i\in[\epsilon]})$, it holds that $\Pr[(\vec{b}_1 = \vec{b}_2 \neq \bot) \wedge (\mathsf{Commit}(\mathsf{pp}, \vec{b}_1) = \mathsf{com})] = 1 - \mathsf{negl}(\lambda)$;

Realizing data availability sampling. Data availability sampling, as introduced in [7, 21], can be realized in many ways. We propose a simple scheme for demonstration. One can use a Linear-Map Vector Commitment (LMVC) [12]: Commit is simply the commitment procedure over an input $\vec{b} \in \{0,1\}^\beta$ (for some $\beta > 0$). Then, $\mathsf{Sampling}$ uniformly samples $R \leftarrow_\$ \mathbb{Z}_q^{\beta\times e}$ for some prime $q = O(\lambda)$, and computes $\vec{b} \times R$. $\mathsf{Reconstruct}$ is also easy: if there are $\epsilon = \beta/e$ samples, Gaussian elimination can be used to reconstruct $\vec{b}$.

[6]While we do rely on DAS, we design a way for agreement without the need to recover the whole block in the permissionless setting (since as discussed in Section 2, block recovery is costly in practice).

Lastly, Vrfy can be done using the LMVC verification. Other prior works like KZG [25], Bulletproofs [8], and so on, can all be used to realize such functionality.

Since we use it as a black box, we can use any construction that satisfies the properties above (and we give a concrete instantiation in section 6.1).

3.2 State Machine Replication Protocol

In this section, we briefly recall the definition of a State Machine Replication (SMR) protocol, which we will also use as a primitive in building our protocol. Each participant in the SMR protocol holds a (local) view of a *chain of blocks*.[7] In each round, the honest participant adds a new block to its chain, where the block is prepared by one of the participants. The two main properties this chain of blocks must satisfy are liveness and consistency. Intuitively, liveness means that any data would be included in the chain if it has already been provided to all the honest parties for a sufficiently long time. Consistency, on the other hand, says that two honest parties will always have two local views where one party's view is a prefix of the other. Additionally, their local views will only ever grow and the added blocks are never edited. Formally, they are defined as follows.

- **Liveness**: If a data blob $\vec{b}$ is posted to an SMR protocol, then for each honest participant holding a view of the chain of blocks $\mathcal{C}$, for each round afterward, if the block is prepared by an honest participant, $\vec{b}$ appears in $\mathcal{C}$.
- **Consistency**: For any two honest parties holding two chains $\mathcal{C}_1, \mathcal{C}_2$ respectively at rounds $r_1 \leq r_2$, it holds that $\mathcal{C}_1$ is a prefix of $\mathcal{C}_2$.

One additional property we care about is *Chain quality* $Q \in [0, 1)$, which we define to be the probability that a block is proposed by a non-honest participant. This probability is over the randomness used to prepare each block and is independent for each round. The smaller Q is, the better the chain is, as the more blocks are prepared by honest participants.

Lastly, for an SMR protocol Π, we define an additional function $\Pi.\mathsf{height}$ which, if called by participant p, simply outputs the length (height) of $\mathcal{C}$ held by p.

The sleepy model. We want our SMR protocol to maintain all the properties above in the sleepy model (i.e., allow for dynamic participation). This model, formally introduced in [30], captures the dynamic participation nature of Bitcoin, where the participants are free to enter and exit the network at any time, and the only assumption needed is that the malicious fraction of total active nodes does not exceed some (predetermined) upper bound $f < 1/2$. To model this, the adversary is allowed to put any honest participant to sleep (make such participant inactive), as long as the constraint on the fraction of malicious participants is still satisfied.

3.3 Digital Signatures

We use the standard digital signature primitive, and here we briefly recall what it does. Essentially, a digital signature allows one to sign an (arbitrarily long) message using their secret key sk, and everyone with the corresponding public key pk can verify the signature. The signature cannot be forged (i.e., only the owner of sk can produce a valid signature). Since digital signature is very standard and widely studied and used in many works (e.g., [15, 13, 23]), we omit the formal definitions for simplicity.

4 Definition and Model

In blockchain systems, a block can be seen as a blob of data that we would like every party to agree on. Currently, there are many protocols that allow a set of parties to agree on a blob of data without scaling

[7]Here, the chain contains only the confirmed blocks. For example, for Bitcoin, the chain only contains the block that are γ blocks deep for some $\gamma > 0$ being the blocks needed for confirmation.

the communication linearly in the size of the data being distributed in the authenticated setting with a PKI in the fixed participation model [26, 33]. However, in the unauthenticated setting, under the dynamic participation model, the best-known solution is to employ Bitcoin [27] and embed all the data in each block. This requires a broadcast to all participants, and the communication cost is linear in the data blob size. Our goal is to devise a protocol that works in the unauthenticated and dynamic participation model, and allows parties to agree on a blob of data using communication sublinear in the size of the blob.

Instead of downloading the whole data blob to be agreed upon, we would like each party to download only a (small) fraction of the data. This can be very useful to use on top of protocols such as Bitcoin where security and latency of communication are correlated (the block interval time necessary for security increases with the communication cost each party needs to have for Bitcoin, and thus the throughput cannot be improved by simply enlarging the block size). Using our protocol one can hope to scale the amount of data participants agree on without requiring an increase in block interval time, scaling the throughput of the protocol. Before introducing our protocol, we formally define the primitive we realize and the model we work in.

4.1 Verifiable Information Dispersal with Strong Termination

In this section, we present the primitive Verifiable Information Dispersal with Strong Termination (VID-ST). As aforementioned, this primitive is closely related to the Verifiable Information Dispersal (VID) primitive [11, 22, 36, 28], with one key difference. We first present our definition, and then discuss how it differs from the definitions studied in prior works.

Definition 4.1. *A Verifiable Information Dispersal with Strong Termination (VID-ST) scheme has three procedures:*

- $\mathsf{Disperse}(\vec{b}) \to \mathsf{com}$*: A deterministic algorithm invoked by the proposer of a blob* $\vec{b} \in \{0,1\}^*$ *to initiate dispersal of a blob* $\vec{b}$*. This algorithm outputs a blob commitment* com*.*

- $\mathsf{Verify}(\mathsf{com}) \to \{0,1\}$*: A randomized interactive protocol invoked by the validators to determine availability of a blob that was dispersed. It takes as input the commitment* com *of the blob being proposed and outputs a bit* b *denoting whether a blob is available.*

- $\mathsf{Retrieve}(\mathsf{com}) \to \vec{b}$*: An interactive protocol invoked by a client. It outputs a data blob* $\vec{b} \in \{0,1\}^*$*.*

Our efficient information dispersal protocol must satisfy the following properties. At a high-level, it has to satisfy (1) termination: if a proposer proposes a block, any participant can eventually decide if the blob is available; furthermore, if the proposer is honest, other honest proposers should view it as available; (2) agreement: every honest participant holds the same view[8]; (3) availability: every block that is viewed as available must be reconstructable by an honest participant[9]; (4) correctness: the reconstructed blob must be the same as the one committed when proposed. These properties are formalized as follows.

- **Termination:** If any proposer invokes $\mathsf{com} \leftarrow \mathsf{Disperse}(\vec{b})$, then every honest client that invokes $\mathsf{Verify}(\mathsf{com})$ eventually gets an output. Further, if the proposer is honest, then the output of $\mathsf{Verify}(\mathsf{com}) = 1$.

- **Agreement:** If an honest client calls $\mathsf{Verify}(\mathsf{com})$ and gets output b, all other honest clients calling $\mathsf{Verify}(\mathsf{com})$ also gets b.

[8]Note that this property is *not* implied by termination, as termination does not specify the output if the proposer is malicious. Thus, a protocol with only termination may have a different view on a maliciously proposed block. With "agreement", if the proposer is malicious, then honest participants all hold the same view, but note that this view can be either "available" or "unavailable".

[9]This means that if the blob is proposed by a malicious proposer, if the honest participants agree that it is available, then, they should later be able to reconstruct it even if the malicious proposer refuses to help.

- **Availability:** If an honest client has output $\mathsf{Verify}(\mathsf{com}) = 1$, then if an honest client invokes $\mathsf{Verify}(\mathsf{com})$, it eventually reconstructs some block $\vec{b}'$.
- **Correctness:** Any two calls to $\mathsf{Retrieve}(\mathsf{com})$ outputting $\vec{b}_1, \vec{b}_2$, where $\mathsf{Verify}(\mathsf{com}) = 1$, satisfy the equality $\vec{b}_1 = \vec{b}_2$. Furthermore, if com is produced by an *honest* client using $\mathsf{Disperse}(\vec{b})$, then $\vec{b} = \vec{b}_1$.

Note that the difference between the properties required in our scheme and the properties required in previous definitions of VID boil down to a *stronger termination property.* In our scenario, we require parties to terminate, in the sense of getting a result from $\mathsf{Verify}(\mathsf{com})$ instead of being stuck in an undecided state, even when the proposer is malicious. This is not necessary in classic VID protocols because usually there is a Byzantine Agreement (BA) round after the VID which forces parties to stop waiting indefinitely for an undecided VID result. However, our protocol is intended to be bootstrapped on top of an existing SMR protocol where the set of participants is not necessarily well defined, so our protocol itself needs to provide the strong termination property rather than rely on a round of BA after.

4.2 Security Model

In this work, we work under the following model.

A synchronous network. There exists a known time bound $\Delta > 0$ such that any message over the network is received by any other party within Δ units of time after delivery. This is the model considered in standard Bitcoin analyses [18, 32, 19] and the model we consider in this work as well.

All participants share the same computational power. This is for simplicity of analysis, we can always switch the model where participants have different amounts of power to this one by treating a single powerful participant as a group of multiple entities with smaller computational power.

Honest majority. We require there are at most $0 \leq f < 1/2$ fraction of malicious participants. This is a commonly used assumption and is also used in Bitcoin [27].

The sleepy model. As mentioned, we also work in the sleepy model [30]. At any point of time t, the number of participants online is $n_t > 0$, as long as the number of malicious participants is bounded by $f \cdot n_t$.

The total computational power in the system does not fluctuate too much within a short period of time. In particular, we require two conditions to hold: (1) the computational power does not increase by much within a short period of time; (2) the computational power does not decrease by much within a short period of time. These are *also* needed by Bitcoin as explained below. More formally:

- (Increase) there exists some $T_i, T_i' > 0$, such that for any $t' \leq T_i'$, $t \geq T_i$, and $s > 0$, let M be the average number of online participants in time $[s, s+t]$, it holds that the average number of online participants over time $[s+t+1, s+t+t']$ is at most $c_i \cdot M$ for some known constant $c_i \geq 1$.

  This assumption is also used in standard security analyses of Bitcoin [32, 19], since Bitcoin security can be violated if this constraint is violated. For example, if at some point, the computational power grows to 100x larger than the average computational power over 45 the previous blocks, then, it means that the adversary, controlling 45% of this amplified computational power, can simply fork a new chain from 45 blocks ago and create 45 new blocks. The computational power it takes is roughly the same as producing a new block. Thus, with this attack, the adversary can easily produce a double-spending attack, which breaks the consistency of Bitcoin. This is forced in the current implementation of Bitcoin as well: the mining difficulty can only change by at most a factor of 4 every 26 weeks [17].

- (Decrease) there exists some $T_d, T_d' > 0$, such that for any $t' \leq T_d'$, $t \geq T_d$, and $s > 0$, let M be the average number of online participants within time $[s, s+t]$, it holds that the average number of online participants in time $[s+t+1, s+t+t']$ is at most M/c_d for some known constant $c_d \geq 1$.

  Similarly, this assumption is also needed for Bitcoin for a similar reason. As the attack above, the adversary could privately mine a long chain using some low difficulty level (since it knows that the computational power would decrease by a lot and thus the difficulty level would decrease), thus creating a long fork.

On the other hand, we believe this assumption is indeed practically reasonable as well: by analysis of the Bitcoin protocol, we found that the mining hash rate never oscillates too much (empirical analysis of hashing rate charts showed no increase greater than 2x for every 26 weeks since 2019 [16]). Thus, we believe having this assumption is reasonable and matches the security guarantee of Bitcoin.

Existance of reliable participants. We define one additional type of participant: a *reliable* participant. We say an honest participant is *reliable* if the participant comes back online eventually, after going offline, although the time may be unknown. Our assumption requires that there are at least some reliable parties exist (note that these parties also do not need to be authenticated, thus still permissionless). Similar to all prior VID constructions [11, 22, 36, 28], this is to guarantee the availability property of our construction.

We argue that this is a *necessary* condition to save communication. In particular, if there is no reliable participant, then it is information-theoretically impossible to guarantee availability other than letting everyone hold the blob (i.e., the protocol must have $\Omega(\beta)$ communication for per data blob of β bits), and thus cannot reduce the communication cost. The reasoning is as follows: if each participant holds $b \leq \beta$ bits of information, by the sleepy model, the adversary can simply put all of the participants to offline except for $\lceil \beta/b - 1 \rceil$ participants. Since none of the participants is reliable, this blob is never recoverable (all the participants online together hold $< \beta$ bits of information about this blob). Similarly, if there are only $O(1)$ number of reliable participants, the saving can at most be $O(1)$. Since this is the *only* additional requirement we need compared to Bitcoin, we believe our model is minimal and optimal to achieve Bitcoin rollups with sublinear communication.[10]

5 Our Scheme

In this section, we discuss our construction.

5.1 Failed Strawmen

Before giving our protocol, we go through two illustrative failed strawmen, to bring to light the intricacies of building such a protocol.

Strawman 1: Random sampling and rejecting a blob if sampling fails. The first idea that comes to mind is to have the proposer send the blob commitment to the SMR chain, and subsequently, every participant (validator) sends the proposer a sampling request of the data. If the request is served, then the validator accepts this blob as available and works on extending this chain, and if it fails, it takes this blob as unavailable and works on proposing a new blob. This fails to satisfy our agreement property since an adversary can propose a blob, and selectively reply to only a subset of honest validators, therefore causing a split view.

Strawman 2: Honest validators broadcast sampling result. This idea builds on the previously proposed strawman, except validators don't decide only based on the result of their own sample request. Instead, honest participants broadcast their sampling results to others (i.e., a vote of yes or no on whether the sampling was successful) and wait for a certain amount of time for others' votes. Note that to avoid simple Sybil attacks, each vote should be accompanied by some proof of identity. For simplicity, let us assume this is already satisfied for this strawman. After receiving all the other validators' votes, each party decides whether this block by counting whether there are at least τ yes votes, for some threshold $\tau \geq 1$. Ideally, if the adversary can at most produce F votes and as long as $\tau > F + \epsilon$, where ϵ is the minimum number of samples needed to recover the whole blob, this should work.

Although this is closer to the final idea we introduce, there is still the issue of agreement. An adversary can potentially send samples to only a fraction of validators right before the decision deadline (such that they do not have time to propagate the votes they receive to other honest validators), and cause only a subset of honest validators to confirm a blob (since those who receive the votes have more than τ yes votes and those who do not receive have less than τ yes votes), again breaching the agreement property.

[10]See Remark A.2 for a discussion on some additional discussion on reliable nodes.

5.2 Our Protocol

To build our scheme, we first specify some public parameters and clarify some terms:

1. λ : the security parameter for the scheme.
2. ϵ: number of samples enough to recover the entire data blob (see Section 3.1).
3. pp: public parameter of the data availability sampling protocol (generated via $\mathsf{Setup}(\lambda, \epsilon)$).
4. $f < 1/2$: upper bound on the fraction of the malicious participants online that our protocol is robust against.
5. An underlying SMR protocol Π (e.g., Bitcoin).
6. Q: the chain quality of Π.
7. One-unit-of-work denotes a unit of work done when some difficulty level is specified. In other words, the proof-of-work puzzle has some difficulty level, and one unit of work is finding one solution to the puzzle under that difficulty level. Proof-of-one-unit-of-work is the proof for this one-unit-of-work (i.e., answer to the puzzle).
8. β: data blob length
9. u: the minimum number of votes to confirm a blob.
10. k: the minimum number of blocks to confirm a data blob.
11. r: the lower bound of the fraction of the reliable participants online.
12. T: the threshold on the number of votes, set according to r, ϵ

5.2.1 The Basic Idea

Our first key observation is that, as shown in the strawmen above, the major issue is that participants who successfully sample cannot agree with the participants who fail. Thus, instead of letting them decide locally, all the participants should agree on this by using the underlying SMR protocol.

In more detail, the proposer of a data blob $\vec{b}$ first adds a commitment $\mathsf{com} \leftarrow \mathsf{Commit}(\mathsf{pp}, \vec{b})$ to Π as a transaction. This will allow parties to check if the sampling reply they receive is consistent with the commitment proposed.

Then, we need to ensure that (1) when an honest participant proposes a blob, all honest participants will see it as available, and (2) honest participants retrieving a blob deemed as available, pertaining to the same commitment com always reconstruct the same data blob $\vec{b}$ (with the help of a set of honest participants).

Similar to the strawmen above, our approach is for participants (validators) to request random samples of blob commitments included in Π. Parties then *vote* on the availability of the block by including a proof-of-one-unit-of-work, along with a vote for *yes*, if the sample(s) it requested were answered correctly by the proposer (which it can verify using the commitment to $\vec{b}$ in Π), or *no* otherwise. The validators post their votes to Π.

A valid vote includes a proof-of-work for one unit of work in order to protect against Sybil attacks. After a window of k blocks of voting, validators decide on whether a blob is available or not by counting the votes and checking whether the number of *yes* votes passes a certain threshold with respect to the *no* votes. The intuition is that given enough time, we will get a good representation of honest validators' view of $\vec{b}$. Since the majority of mining power is honest, available data blobs will always have a significantly greater number of *yes* votes than *no* votes, and vice versa. Then, the validators simply mark a blob as available if the yes/no vote threshold is met, and mark the blob as not available otherwise. After the confirmation, a new blob can be proposed.

Two remaining issues. While this scheme seems to work, there are two major remaining issues.

- **Multiple blobs and proposers.** First, it is not immediately clear how to support multiple blobs being validated concurrently without compromising security (splitting the voting power across different commitments compromises security, on which we expand below). However, supporting validating multiple commitments concurrently as such would greatly increase the throughput of the scheme. Thus, we propose a solution that allows for this without any additional cost or any security compromise.
- **An arbitrarily small set of validators.** Second, we must deal with the fact that the total number of participants (validators) online can be arbitrarily small in the standard sleepy model. If, say, there are only 3 validators online (which is the minimum number of validators online with at least one malicious validator, as $f < 1/2$), then even after k blocks, to make sure that the votes within k blocks are representative enough, we need these validators to produce a huge amount of votes. A simple solution is to assume that there are at least $\bar{N}$ participants online for some $\bar{N} > 3$, such that the parameters can be set according to $\bar{N}$. However, even with such relaxation in the sleepy model, $\bar{N}$ needs to be set very conservatively which blows up the communication and latency of our scheme.

  Making an aggressive assumption on the number of participants online (i.e., large $\bar{N}$) can instead lead to breaches in our security properties. Note that if that threshold is not met and thereby we don't get enough votes, we lose our Termination and Availability property (since the malicious party could dominate the voting result, thus making an available blob be viewed as unavailable or making an unavailable blob be viewed as available). Therefore, we want a scheme that is secure for any number of participants online and whose communication can adapt to the number of parties currently online (instead of a conservative bound).

5.2.2 Our Solution

We now discuss how to address these two issues.

Pipelining. As mentioned, the natural question is whether we can pipeline. Naively doing it (i.e., producing the vote independently for each of the previous k commitments in k blocks) does not work for the following reason: an honest validator needs to split their computational power over k blobs, while a malicious validator can spend all its work on a single blob commitment. In this case, the malicious validators can convince the honest ones that an unavailable blob is available (or conversely, make a blob from an honest proposer be viewed as unavailable) since they produce k times a larger amount of fake votes and thus can dominate the voting result.

However, our observation is that the works do not need to be separated for multiple votes: instead of producing one proof-of-work per commitment, each validator generates a single proof-of-work and append the votes for *all* the k blobs to the proof. To avoid the votes being tampered, we propose to have the proof-of-work associated with an ephemeral signature public key (i.e., the puzzle answer contains an ephemeral signature public key). The appended votes are signed by the secret key corresponding to this public key. Then, everyone verifies the signature and makes sure that the votes are indeed valid.

Remark 5.1. *One may wonder if the underlying SMR also requires computational power (e.g., Bitcoin), whether this separation of computational power incurs similar security issues. Luckily, we can simply merge the two proof-of-work. For example, we can set the difficulty of our puzzle easier than the one by Bitcoin but based on the same hash function. Then, when mining Bitcoin, participants can find the solution to our puzzle without separating the computational power. A similar argument holds if the difficulty level we need is harder.*

Setting a lower bound for the total number of votes. To avoid the second, more delicate issue, we propose that instead of only waiting for k blocks, we also wait for u total votes. In other words, we decide on a block only if there are at least k blocks *and* u votes in these $\geq k$ blocks (otherwise we keep waiting for the next block).

This addresses the issue as follows: we now can set u, k to be large enough to be representative of the honest participants' opinions (we detail the exact method for picking these parameters in Section 5.2.3), instead of having k blocks containing an unknown number of votes dependent on the number of online users. In this case, since u is independent of the number of online validators, the issue above does not exist.

The only question left now is how to set the difficulty for one-unit-of-work. Fortunately, this is easy: we can first *guess* the number of participants online. Let us denote our guess for the number of participants online as $\hat{N}$. We then set the difficulty level to be such that $\hat{N}$ participants (in expectation) can finish u units of work in k blocks. One essential observation is that now the accuracy of $\hat{N}$ *does not* affect the security: no matter $N > \hat{N}$ or $N < \hat{N}$, there are always enough votes (and blocks) such that the votes represent the honest participants' opinions. Therefore, $\hat{N}$ only affects the efficiency (i.e., if $\hat{N} > N$, it takes more than k blocks to finish; and if $\hat{N} < N$, k blocks produce more than u of votes).

However, since now $\hat{N}$ does not affect the security, we do not need to be conservative. Instead, we can simply adjust $\hat{N}$ (and consequently the difficulty level) dynamically according to the computational power for the recent blocks, just like what Bitcoin does for every 2016 blocks.[11] In this case, $\hat{N}$ has a relatively accurate representation of N, thus allowing the efficiency to be close to optimal.

We present our construction formally in Figure 2.

5.2.3 Setting u, k, and the difficulty level

So far, we have not discussed how we should set u, k, T. Note that we need to set the parameters so as to guarantee that a blob which was voted available can always be correctly retrieved by a set of honest parties and that all the honestly proposed blobs are always confirmed.

Setting the threshold T. First, we need to set T such that if T samples are held by the honest parties, then ϵ samples are held by the reliable parties (recall that reliable participants come back online eventually.) In expectation, $R_1 := \frac{rT}{1-f}$ samples are held by reliable parities. Thus, by Chernoff bound, except for $\mathsf{negl}(R_1)$ probability, we have $R = R_1 - \log^2(R_1)$ samples are held by reliable parties. We thus simply need to set $R = \epsilon$. After setting this, we can set u, k accordingly.

Setting u, k, case 1: Protocol terminates within exactly k blocks with u votes cast. We start with the simplest case: consider the case that by the k-th block, there are exactly u votes.

Let h_{com} be the height of the block in Π at which a commitment com to data blob $\vec{b}$ appears. Also, let $k_f \in \{0, \ldots, k\}$ be the size of the consecutive chain of blocks controlled by the adversary at the final blocks of the k interval determining the votes towards commitment com; in other words, the block with height $h_{\mathsf{com}} + k - k_f$ represents the last block in Π in which this block was prepared by an honest participant voting for com. For any $k_f \in \{0, \ldots, k\}$, the probability that this happens is $1 - Q^{k_f}$ (recall that Q is the chain quality of Π, which is the probability that a block is proposed by a malicious participant in Π). Similarly, k_f is also the number of blocks that the adversary can keep the block containing the blob commitment com private, except for $1 - Q^{k_f}$ probability (to produce votes privately without allowing other participants to produce votes).

Furthermore, by our assumption in Section 4.2, the average computational power of the last k_f blocks is at most c_i times larger than it is in the first $k - k_f$ blocks; and the average computational power of the k_f blocks that the adversary privately produces votes can at most be c_d times larger than the first $k - k_f$ blocks after the block .[13] Thus, in the worst case (i.e., (1) the last k_f blocks are all malicious, and the increase in computational power reaches c_i; and (2) the adverasry keeps the block private for k_f blocks, and the decrease in computational power reaches c_d), among the u votes, in expectation, $F_1 = f \cdot (c \cdot k_f + k - k_f) \cdot u/k$ votes are from the malicious party, where $c := c_i + c_d$. By Chernoff bound, at most $F = F_1 + \log^2(F_1)$ votes are produced by the malicious party, except with $\mathsf{negl}(F_1)$ probability. Thus, there are at least $H = u - F$

[11] Note that such dynamic adjustment is *not* possible if $\hat{N}$ affects security: recall that in the sleepy model, the adversary can put participants into sleep and thus let $N \ll \hat{N}$, which causes the issue we mentioned above.

[12] Again, if the underlying Π is proof-of-work based, this can be merged with the underlying Π as discussed in section 5.2.2.

[13] Note that we need to have T_i, T_i', T_d, T_d' satisfy that (1) the time for producing k_f blocks is smaller than T_i', T_d' and (2) the time for producing $k - k_f$ blocks is greater than T_i, T_d. Then, by our model assumption, the statement in the main body holds. These conditions are specified in Theorem 5.2.

Global operation. Each participant holds some ephemeral signature key pairs (which can be destroyed and regenerated after a single use, or kept used until the participant wants to destroy it). While online, each participant keeps doing one unit of work with respect to (1) the ephemeral signature public key pk (who holds corresponding secret key sk), (2) the current blockchain head (i.e., the latest block header H). The one-unit work is to find a prefix x such that the hash of "$x||\mathsf{pk}||H$" is smaller than some threshold and the proof $\pi := (x, \mathsf{pk}, r)$ given some difficulty level, specified how to set in Section 5.2.3.[12]

$\mathsf{Disperse}(\vec{b})$:

1. Post $\mathsf{com} \leftarrow \mathsf{Commit}(\mathsf{pp}, \vec{b})$ to Π.
2. Output com.

$\mathsf{Verify}(\mathsf{com})$:

1. Let h_{com} denote height of the block in which com was included (in Π).
2. Let Y denote number of *valid* yes votes for com between blocks h_{com} and $\Pi.\mathsf{height}$, and O be equivalently defined for no votes. A vote is valid if (1) the proof-of-one-unit-of-work is checked against $x||\mathsf{pk}||H$ and H appears after (or is the same as) h_{com} (i.e., the header used is the block com appears or the blocks after); (2) the signatures are valid against pk. See below for what makes a valid vote a "yes" or "no" vote.
3. **While** $\Pi.\mathsf{height} < h_{\mathsf{com}} + k$ or $Y + O < u$ **do** :
   - When starting each unit of work, send to the proposer of com some randomness s.
   - Proposer replies with $R \leftarrow \mathsf{Sampling}(\mathsf{pp}, \vec{b}; s)$.
   - Client runs $b \leftarrow \mathsf{Verify}(\mathsf{pp}, \mathsf{com}, R)$ and posts a signed vote $V = \{\mathsf{com}, \pi, b\}$ (signed by the corresponding secret key sk) to Π. If a reply is not received by π is finished (i.e., the one unit of work is done), $b \leftarrow 0$. Here if $b = 1$, it is a "yes" vote and otherwise a "no" vote.
   - Update Y, O accordingly if a new block appears.
4. If $\Pi.\mathsf{height} \geq k$ and the u-th vote is in the last block (i.e., in the k-th block or the blocks after), let Y' be the number of yes votes among the first u votes (note the number of votes is always greater than or equal to u), and O' be the number of no votes among the first u votes.
5. If $\Pi.\mathsf{height} = k$ and the u-th vote is *not* in the k-th block, let Y' be the number of yes votes among *all* the votes in these k blocks, and O' be the number of no votes among *all* the votes (note that in this case, there are more than u votes and we count them all).
6. Output $Y' \geq O' + T$, where $T > 0$ is some threshold dependent on ϵ (the number of samples needed to reconstruct a blob) and r (the fraction of reliable participants), specified in detail in section 5.2.3.

$\mathsf{Retrieve}(\mathsf{com})$:

1. The participant calling this $\mathsf{Retrieve}$ procedure broadcasts com.
2. Parties who hold R being a reply from $\mathsf{Sampling}(\mathsf{pp}, \vec{b})$ requested against com in the Verify protocol send R back.
3. Upon receiving ϵ unique replies $\{R_j\}_{j\in[\epsilon]}$ such that $\mathsf{Vrfy}(\mathsf{pp}, \mathsf{com}, R_j) = 1$, the party runs $\vec{b} \leftarrow \mathsf{Reconstruct}(\mathsf{pp}, \{R_j\}_{j\in[\epsilon]})$.
4. Output $\vec{b}$.

Figure 2: Our Verifiable Information Dispersal with Strong Termination protocol.

honest votes. We hence need to set u, k such that $H \geq F + T$. Note that setting so is always possible since $T, c_i, c_d, Q < 1, f < 1/2$ are all known constants, which are the only variables that determine u and k.[14]

Setting u, k, case 2: Protocol terminates after $\kappa > k$ blocks. This case will happen whenever after k blocks, we have not reached u votes. This means that by the time we get u votes, there have been $\kappa \geq k$ blocks (recall that we only count the first u votes in these $\kappa \geq k$ blocks, so we can assume that exactly u votes were cast), for some variable κ. Thus, we instead need to define F_1, F, H as functions over the variable κ instead of a constant k. In other words, $F_1(\kappa) := f \cdot (c \cdot k_f + \kappa - k_f) \cdot u/\kappa = f + \gamma/\kappa$ where $\gamma := f \cdot (c \cdot k_f - k_f) \cdot u$ being a constant. Thus $F(\kappa) := F_1(\kappa) + \log^2(F_1(\kappa))$ and $H(\kappa) := \kappa - F(\kappa)$.

Then, let $D(\kappa) = H(\kappa) - F(\kappa)$ be the difference between honest votes and malicious votes. We have $D(\kappa) = H(\kappa) - F(\kappa) = u - 2F(\kappa) = u - 2(f + \gamma/\kappa + \log^2(f + \gamma/\kappa))$. Then computing its derivative gives $\frac{\partial D}{\partial \kappa} = \frac{2}{\kappa^2} + \frac{2\gamma \log(\frac{c+\kappa}{\kappa})}{\gamma\kappa + \kappa^2}$. Note that since $\gamma \geq 0$ (as $c \geq 1$), we have $\partial D/\partial \kappa > 0$ for any $\kappa > 0$. Thus, $D(\kappa)$ is a monotonically increasing function. By case 1, we know that $D(k) = H(k) - F(k) = H - F \geq T$. Thus, for any $\kappa \geq k$, it holds that $D(\kappa) \geq D(k) \geq T$. Therefore, it does not matter whether the verification process finishes in more than k rounds.

Setting u, k, case 3: Protocol terminates within k blocks with $\mu > u$ votes cast. The remaining case to consider is when we have reached u votes *before* k blocks. This means, by our protocol, by the k-th block, the validators stop and count all the votes in these k blocks, which contain $\mu > u$ votes for some variable μ. Again, we now define F_1, F, H over μ instead of the constant u. I.e., $F_1(\mu) := f \cdot (c \cdot k_f + k - k_f) \cdot \mu/k = \gamma' \cdot \mu$, where $\gamma' := \frac{f \cdot (c \cdot k_f + k - k_f)}{k}$ being a constant. Thus $F(\mu) := F_1(\mu) + \log^2(F_1(\mu))$ and $H(\mu) := \mu - F(\mu)$ Similarly, we define the difference function $D(\mu) := H(\mu) - F(\mu) = \mu - 2(\gamma'\mu + \log^2(\gamma'\mu))$.

Then, we take the derivative $\frac{\partial D}{\partial \mu} = 1 - 2\gamma' + \frac{4 \log(\frac{1}{\gamma'\mu})}{\mu}$. Now, as long as $\gamma' < 1/2$, it is easy to see that the derivative is positive for any $\mu > 0$. If we can argue that the derivative of our $D(\cdot)$ function is positive, then it follows that for any $\mu > u$, $D(\mu) \geq D(u) \geq T$ and therefore conclude we have enough samples even when finishing with more than μ votes. Going back to what we said before, to argue this, we only need to argue $\gamma' < 1/2$ is indeed true.

We argue this by contrapositive. Note that, if $\gamma' \geq 1/2$, it means that $c \cdot k_f + k - k_f > k$, which means $F_1(u) = \gamma' \cdot u \geq u/2$, and thus $H(u) = u - F(u) = u - F_1(u) - \log^2(F_1(u)) < u - F_1(u) \leq u/2 \leq F_1(u) < F(u)$. However, since we set u, k such that $H(u) > F(u) + T$ (the condition we set in case 1), it is impossible that $\gamma' \geq 1/2$. Thus, $D(\mu)$ is also a monotonically increasing function. Therefore, we conclude that it also does not matter if we reach u votes before k blocks. With these, we can see that we only need to set u, k such that $H > F + T$ according to case 1 above.

Setting the difficulty level. Lastly, we specify what it means by one-unit-of-work, which is done by setting the difficulty level accordingly. To set this, as mentioned, we first guess some $\hat{N}$, which is the average number of participants online. Then, the difficulty level is set such that $\hat{N}$ participants, in expectation, finish u units of work by k blocks. Note that as analyzed above, the accuracy of this guess does not affect any requisite property.[15] Thus, as mentioned, we can dynamically adjust the difficulty level such that $\hat{N}$ can be close to N. More formally, let d be the expected units of work a node can do for a block. We set $d := u/k/\hat{N}$.

With these parameter settings, we complete our protocol and formalize the theorem as follows, and prove the properties accordingly.

Theorem 5.2. *Our protocol in Figure 2 achieves all the properties in Section 4.1, given the parameter setup for u, k above in our model in Section 4.2, where T is the minimum time Π needs to propose k_f blocks and T' is the maximum time Π needs to propose $k - k_f$ blocks.*

[14] Note that ϵ is also a tunable parameter (i.e., the number of samples needed to recover the blob) which is used to determine T, while c_i, c_d, Q, f, r are all environmental parameters that we cannot control. Later in the evaluation section, when finding the u, k, we tune the ϵ as well. However, for simplicity, we assume ϵ is a fixed constant here, which is w.o.l.g., since it only adds additional freedom and does not hurt feasibility.

[15] Even if it is not guessed correctly, the only effect is that it ends up taking more blocks to finish (if $\hat{N}$ is too large) or having more votes by k blocks (if $\hat{N}$ is too small).

Proof. We prove each property separately.

- (Termination) After any proposer invokes $\mathsf{Disperse}$, its output com would be included to Π by the liveness property of Π. Then, for anyone who calls $\mathsf{Verify}(\mathsf{com})$, after k' blocks from the round that com is included by Π, where k' is the maximum number between k and the number of blocks needed to generate u one-unit-of-work proofs by the honest participants (which is in expectation u/d honestly prepared blocks, since in the worst case there is only one honest validator; and by Chernoff bound, this is bounded by $u/d + \log^2(u/d)$ except for $\mathsf{negl}(u/d)$ probability), it is guaranteed to output a bit.

  Furthermore, if the proposer is honest, it means that all the sampling request $\mathsf{Sampling}$ (represented by a randomness s) from honest samplers are replied. Note that as computed, with the u, k set up above, honest samplers can produce H votes of "yes" except with negligible probability, and the malicious samplers can at most produce $F < H - T$ of "no" votes, and thus all the honest proposers' data blobs are confirmed: i.e., $\mathsf{Verify}(\mathsf{com})$ always outputs 1.

- (Agreement) By the consistency property of Π, every honest parties calling $\mathsf{Verify}(\mathsf{com})$ sees the exact same yes and no votes. Thus, they output the same bit.

- (Availability) Now, if $\mathsf{Verify}(\mathsf{com}) = 1$, it means that for com, there are Y yes votes and O no votes and $Y \geq O + T$. Furthermore, $Y + O = H + F$, and we only need to show that $Y - F > T$ (i.e., even if F votes of the yes votes are malicious, there are still at least T yes votes from the honest samplers and thus ϵ are from reliable participants, who together guarantee data availability). To show this, note that we have $Y + O = F + H > 2F + T$. Then, $Y - F > F - O + T$. If $F \leq O$, $Y - F > T$. On the other hand, $F > O$, then $Y - F > Y - O \geq \epsilon$ as needed.

  Thus, we know that among the Y yes votes, at least T of them are from honest participants and among which, ϵ are from reliable participants. Thus, when calling the $\mathsf{Retrieve}(\mathsf{com})$ protocol, these reliable parties would (eventually) reply when they are online. Thus, $\mathsf{Retrieve}(\mathsf{com})$ outputs some data blob $\vec{b}$.

- (Correctness) Any call to $\mathsf{Retrieve}(\mathsf{com})$ outputs $\vec{b}$ by the argument above, as long as $\mathsf{Verify}(\mathsf{com}) = 1$. Then, by the security definition of data availability sampling (Section 3.1), as long as $\mathsf{Vrfy}(\mathsf{pp}, \mathsf{com}, R_j) = 1$ for all $j \in [\epsilon]$, it holds that $\mathsf{Reconstruct}(\mathsf{pp}, \{R_j\}_{j\in[\epsilon]})$ output the same data blob $\vec{b}'$ for every call. Furthermore, if $\mathsf{com} \leftarrow \mathsf{Disperse}(\vec{b})$ generated by an honest party, it holds that $\mathsf{Commit}(\mathsf{pp}, \vec{b}) = \mathsf{com}$, and thus $\vec{b} = \vec{b}'$.

□

5.2.4 Communication Analysis

Now we analyze the communication cost of our construction. Note that we analyze the communication costs of each participant when a blob reconstruct is *not* necessary (i.e., every honest participant is convinced that the blob is reconstructable but does not need the blob itself). This is also what is analyzed in prior VID works [11, 22, 36, 28]. If a reconstruction is needed for every participant, then information theoretically, the communication per recipient is $\Omega(\beta)$ and cannot be improved.

Suppose there are N participants online (on average).

When $\hat{N} = N$. We start with the case where our guess $\hat{N} = N$. Suppose for each time of "posting an item X on Π", it takes $C_\Pi \cdot |X|$ communication cost (e.g., for Bitcoin, the cost is $C_\Pi = O(N)$, i.e. the cost to broadcast it). For our case, $|X| = O_\lambda(1)$ when posting $\mathsf{Commit}(\mathsf{pp}, \vec{b})$, and it is $O(k)$ when posting the votes. Thus, in total, $|X| = O(k)$. Thus, the expected communication cost for voting per user (per block in Π) is $O(C_\Pi \cdot d \cdot k) = O(C_\Pi \cdot u/\hat{N})$.

Then, the expected communication of sampling (per block in Π) is $O(|R| \cdot d \cdot k) = O(|R| \cdot u/\hat{N})$ (recall that R is the reply from the sampling request and d is the expected number of votes a validator can produce given the guessed $\hat{N}$ online validators). For simplicity, we assume $|R| = O(\beta/\epsilon)$ (as discussed in Section 3.1).

Thus, the total communication per user is $O(\frac{C_\Pi \cdot u}{\hat{N}} + \frac{\beta \cdot u}{\hat{N} \cdot \epsilon})$. As an example, when instantiated with Bitcoin, our cost is $O(u + \frac{\beta \cdot u}{\hat{N} \cdot \epsilon})$, while Bitcoin's cost is β.

When $\hat{N} > N$. When we overestimate N, it means that we need more blocks to finish. However, since the total number of votes is still u, the total communication remains exactly the same. The latency (the number of blocks to wait for) grows from k to $u/\hat{N} \cdot N$.

When $\hat{N} < N$. When we underestimate N, it means after k blocks, we have $\mu > u$ votes. In expectation, $\mu = u/\hat{N} \cdot N$. In this case, the total communication cost becomes $O(\frac{C_\Pi \cdot u \cdot N}{\hat{N}^2} + \frac{\beta \cdot u \cdot N}{\hat{N}^2 \cdot \epsilon})$.

In a nutshell. While these are relatively complicated, as mentioned, the bottom line is that the guessed $\hat{N}$ does not affect the security at all. Therefore, we can dynamically adjust $\hat{N}$. In this case, $\hat{N}$ can be assumed to be an accurate estimation of N (i.e., $\hat{N} = O(N)$). Thus, with this assumption, we can conclude that our communication cost is $O(\frac{C_\Pi \cdot u}{\hat{N}} + \frac{\beta \cdot u}{\hat{N} \cdot \epsilon})$ which is again $O(u + \frac{\beta \cdot u}{\hat{N} \cdot \epsilon})$ when instantiating Π with Bitcoin, compared to β bits for the native cost of Bitcoin.

To further simply the asymptotics, note that since c, f, Q are constants, we can view $u = O(\epsilon)$. Thus, the cost is $O(\epsilon + \frac{\beta}{\hat{N}})$. If we set ϵ to $O(\frac{\beta}{\hat{N}})$, then for any $\hat{N} = \omega(1)$ (i.e., the number of online validators is super-constant in the size of the blob), our communication cost is simply $O(\beta/\hat{N}) = o(\beta)$, thus sublinear in the blob size.

6 Evaluation

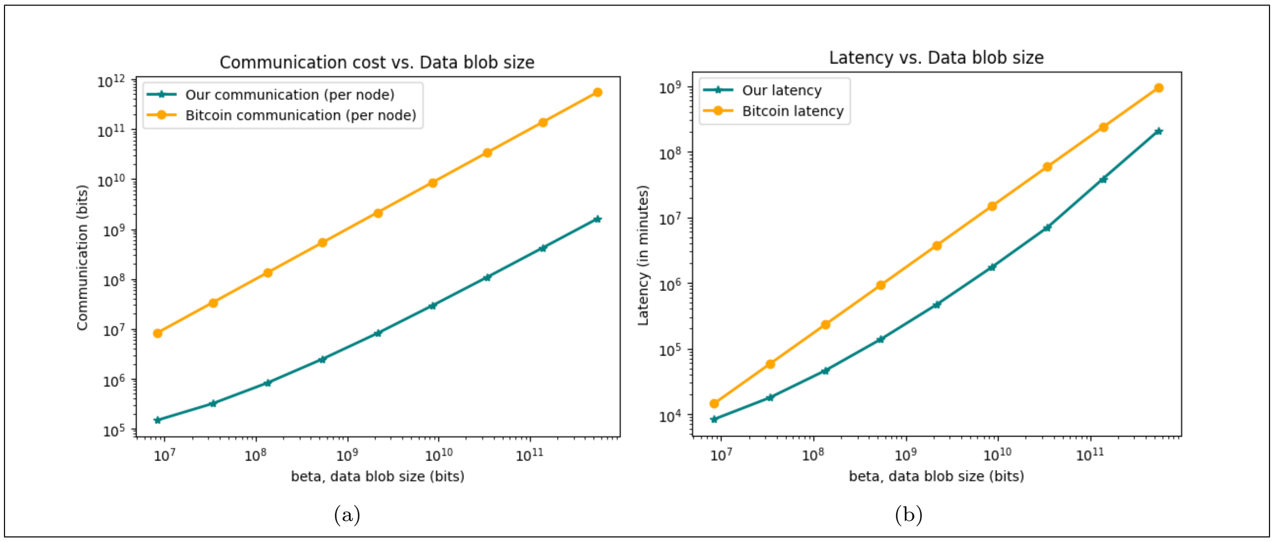


Figure 3: Analytical comparison of communication and latency when β, the blob size, changes.

In this section, we show how our protocol behaves given different parameters in terms of both communication cost (per participant), as calculated in Section 5.2,[16] and latency. For concreteness, we instantiate our underlying SMR protocol with Bitcoin (the numbers are taken according to our implementation discussed below), and compare our results with the current state of Bitcoin. Note that the communication cost (and correspondingly latency) shown in this section is based on the most ideal evaluation. We then discuss our implementation and integration considerations in section 6.1 and show the resulting communication cost considering all the overhead needed to instantiate our protocol on top of Bitcoin without modifying the current Bitcoin implementation in Appendix B.

[16]Note that in the estimation, we view the hidden constant to be 1, and the public key plus signature size to be 512 bits (i.e., sizes for ECDSA). The hidden constant being 1 is reasonable since the vote is a single bit, and the DAS process can indeed achieve $|R| = |\beta|/\epsilon$, using many ways (e.g., our example in Section 3.1 or other ways as discussed in [7]).

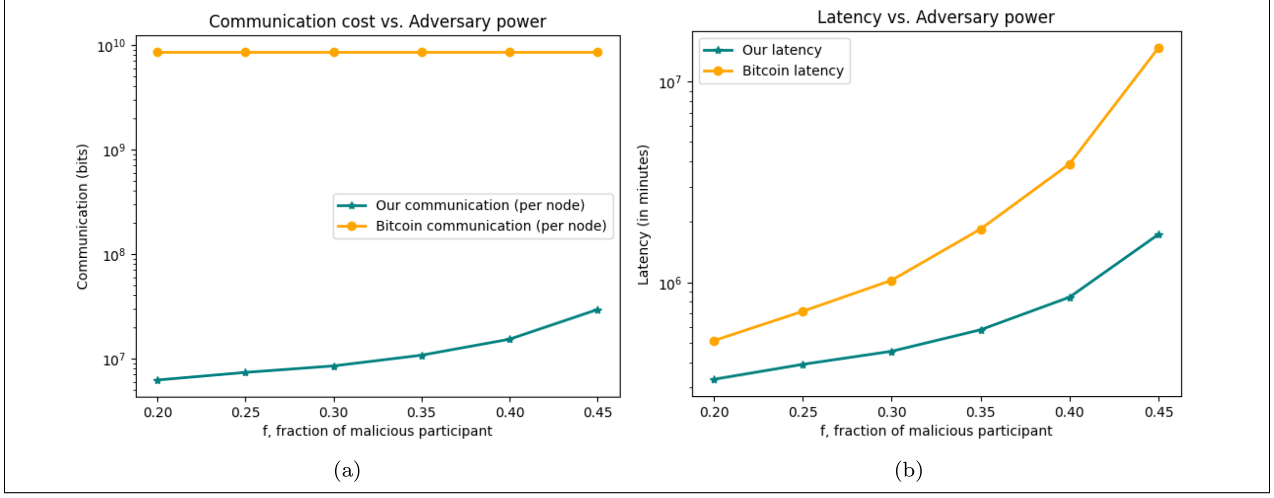


Figure 4: Analytical comparison of communication and latency when f, the fraction of malicious computation power, changes.

Basic parameters. Let us quickly recall what parameters affect the communication cost of our construction.

- $0 < \beta$: number of bits for a data blob
- $0 \le f < 1/2$: the fraction of malicious participants.
- $1 \le c$: the cap on the overall fluctuation of the number of participants (the sum of the increase cap and decrease cap).
- $0 \le Q < 1$: the proportion of maliciously mined blocks in the underlying SMR protocol.
- $0 < e < 1$: the probability that our construction breaks one of the properties in Section 4.1.[17]
- $1 < N$: the real number of participants in the system.[18]
- $0 \le r \le 1 - f$: the fraction of reliable participants.

With these parameters, we show the (expected) communication cost per node and latency of our construction. We choose u and k such that the communication is minimized as discussed in Section 5.2.3.[19] The communication cost can be calculated exactly as discussed in Section 5.2.4. Then, we analyze the latency. Since our protocol is initialized with Bitcoin, the latency is then dependent on the block size (of Bitcoin, which decides the block interval time)[20], and the number of blocks needed to confirm a block. Similarly, we analyze the latency for Bitcoin.[21]

Communication and latency with respect to blob size (β). We first show how our communication cost changes given different β ranging from 2^{23} (i.e., a megabyte) to 2^{39} (i.e., 64 gigabytes). We set $f = 0.45$,

[17]Note that e is dependent on λ, normally set as $e = 2^{-\lambda}$. We use e instead of λ since sometimes we set e to be a non-power-of-two (e.g., $e = 0.001$).

[18]Recall that as mentioned, we can dynamically adjust $\hat{N}$ (the guessed number of participants). Thus, for simplicity, we assume $\hat{N} = N$. However, even if there are some small constant factors off, all our advantages still hold.

[19]Note that ϵ can also be controlled. Thus, we also choose ϵ to minimize the communication.

[20]For our protocol, we view "communication per participant" as the block size. This is a conservation estimation as the size of the votes is smaller than the size of the samples. However, since the sampling also takes bandwidth, we use this conservation estimation to make a fair comparison.

[21]We analyze the latency according to the widely-used Bitcoin setup, 10 minutes of inter-block time for 1MB of block size. And let the inter-block time be proportional to the block size (e.g., for a block size of 2 MB, the inter-block time is 20 min).

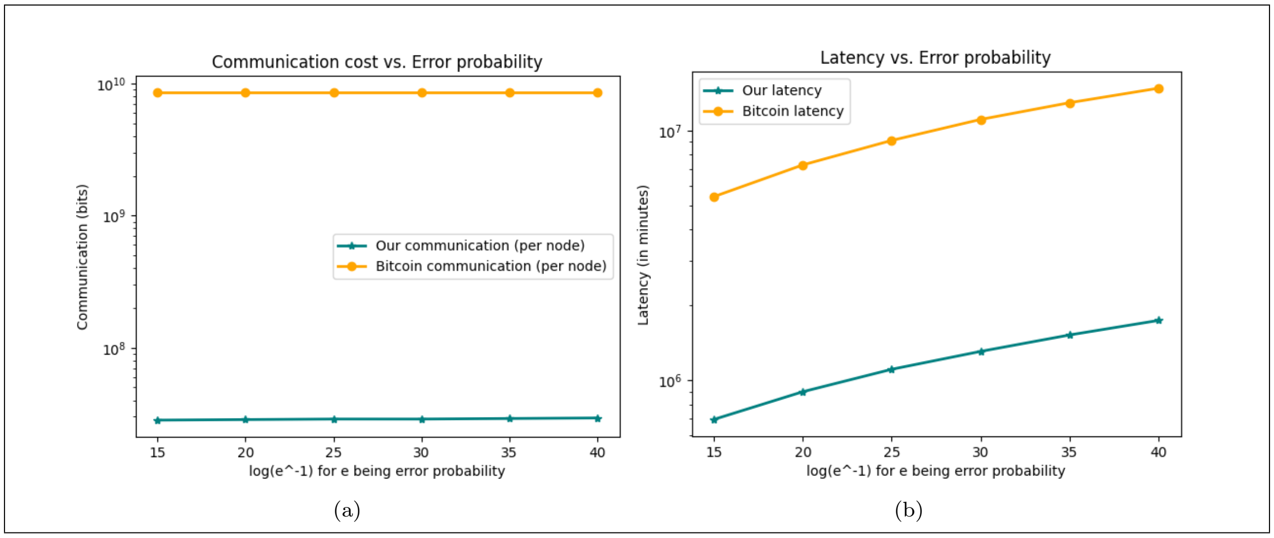


Figure 5: Analytical comparison of communication and latency when e, the error probability, changes.

$e = 2^{-40}$, $c = 6$, $Q = 1/2$, $N = 10,000$, $r = 0.2$. This means that (1) the fraction of adversarial nodes is 0.45; (2) the probability any of our security properties is breached is 2^{-40}; (3)the mining power will not increase or decrease by more than 3x within a short period of time (thus summing to $c = 6$); (4) the chain quality is half; (5) there are 10,000 total participants online; (6) 20% of the partipants are reliable. Note that both x-axis and y-axis are in log-scale. It is easy to show as in Figure 3a, our construction has a much smaller communication cost compared to current Bitcoin for at least around 2 orders of magnitude. As shown, when β grows, our construction shows even better savings and gradually approaches 1000x (three orders of magnitude) smaller communication cost. Furthermore, note that our communication cost is *sublinear* in β: as shown in the graph, the slope of our curve is less than 1 (Bitcoin has a slope of 1), and since our graph is in log-log scale, this means that our curve is sublinear in β.

Similarly for latency (shown in Figure 3b), our latency is about 10x smaller than Bitcoin. The parameters are tuned to minimize communication, so we see more modest latency improvements than communication improvements. If the parameters are picked differently, one can push latency further down to a certain extent at the cost of communication. However, we found the current trade-off to be the most desirable.

Communication cost and latency with respect to the adversarial fraction (f). In Figures 4a and 4b, we show how our communication cost changes given different f ranging from 0.1 to 0.45. We set $\beta = 2^{33}$, $e = 2^{-40}$, $c = 6$, $Q = 1/2$, $N = 10,000$, $r = 0.2$. Our scheme outperforms Bitcoin for all the possible values of f, in terms of both communication cost *and* latency. Note that the y-axis is still in log-scale, and thus our construction remains about 2-3 orders of magnitude better communication cost than Bitcoin for this parameter setup, and about one order of magnitude better in terms of latency.

Communication cost and latency with respect to failure probability (e). In Figures 5a and 5b, we show how our communication cost changes given different e ranging from 2^{-15} to 2^{-40}. We set $\beta = 2^{33}$, $f = 0.45$, $c = 6$, $Q = 1/2$, $N = 10,000$. Similarly, our construction performs better for any e in terms of both communication and latency.

Communication cost and latency with respect to the compute power volatility (c). Recall c is how much volatility in compute power we can support within a given k block window. In Figures 6a and 6b, we show how communication costs change given different c ranging from 2 to 20 (i.e., the increase and decrease are both capped by 10x, summing to 20). We set $\beta = 2^{33}$, $f = 0.45$, $e = 2^{-40}$, $Q = 1/2$, $N = 10,000$, $r = 0.2$. Similarly, our construction's communication and latency increase with c but the fluctuation is small and we preserve an improvement of around three orders of magnitude in communication and one order of magnitude in latency with respect to Bitcoin even while being resilient against computational power fluctuations of up to 10x (and such trend continues with even larger c).

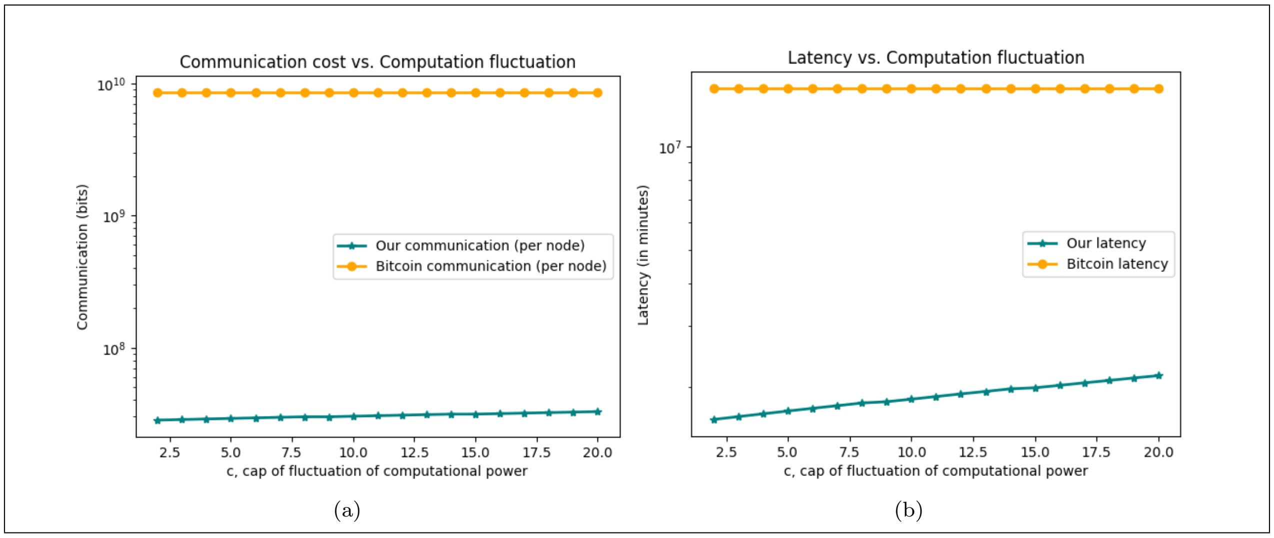


Figure 6: Analytical comparison of communication and latency when c, the cap of computation power volatility, changes.

Communication cost and latency with respect to chain quality (Q). In Figures 7a and 7b, we show how our communication cost and latency change given different Q ranging from 0.1 to 0.8. We set $\beta = 2^{33}$, $f = 0.45$, $e = 2^{-40}$, $c = 6$, $N = 10,000$, $r = 0.2$. Note that communication cost is not greatly affected by chain quality in either scheme. In terms of latency, a diminishing chain quality increases the latency of our scheme, however, even with very low chain quality, our latency is still much smaller than Bitcoin's.

Communication cost and latency with respect to the real number of participants (N). Our communication cost also changes with respect to N, the real number of online participants as shown in Figures 8a and 8b. We set $N = 1,000$ to $1,000,000$. Additionally, we set $\beta = 2^{33}$, $f = 0.45$, $e = 2^{-40}$, $c = 6$, $Q = 1/2$, $r = 0.2$. It is easy to see that our communication and latency both decrease very fastly as N grows, since as mentioned, our scheme shines when the number of participants online is large: recall that our main idea is to have each participant only need to hold a small portion of the data; thus, when the number of participants grow, the size of the portion decreases.

Again, recall that we assume our guessed number of online participants $\hat{N}$, which can be dynamically adjusted, accurately estimates N (i.e., $\hat{N} = N$). If $\hat{N} = \alpha N$ for $\alpha > 1$, we can simply multiply our communication in the figure by α (for $N = 10^6$, this α has to be $> 10^4$ to make our communication cost unfavorable compared to Bitcoin). Similarly, if $\alpha < 1$, we can simply divide our latency in the figure by α (for $N = 10^6$, this α has to be $< 10^{-3}$ to make our latency unfavorable compared to Bitcoin).

Communication cost and latency with respect to the fraction or reliable participants (r). Our communication cost also changes with respect to r, the fraction of reliable participants as shown in Figures 8a and 8b.[22] We set $r = 0.1$ to 0.55 (i.e., $1 - f$ for the f below). Additionally, we set $\beta = 2^{33}$, $f = 0.45$, $e = 2^{-40}$, $c = 6$, $Q = 1/2$, $N = 10,000$. Similar to N, the communication costs and latency also decreases fastly with respect to r. However, even if r is as small as 0.1, our construction still remains more than two orders of magnitude communication advantage compared to Bitcoin, and slightly less than one order of magnitude latency advantage. With larger r, such advantage enlarges quickly.

[22]One thing to note is that the reliable participants can be different and independent for each data blob. In other words, a participant may be reliable only for a particular data blob (block) instead of always being reliable, and this is already sufficient for our construction.In particular, the participant only needs to reply to the reconstruction request to the particular blob(s) that it is reliable against. The only requirement we need is that for each data blob, there is r fraction of participants sampling it being reliable against this blob. This is a weaker assumption than having participants being reliable throughout the entire blockchain.

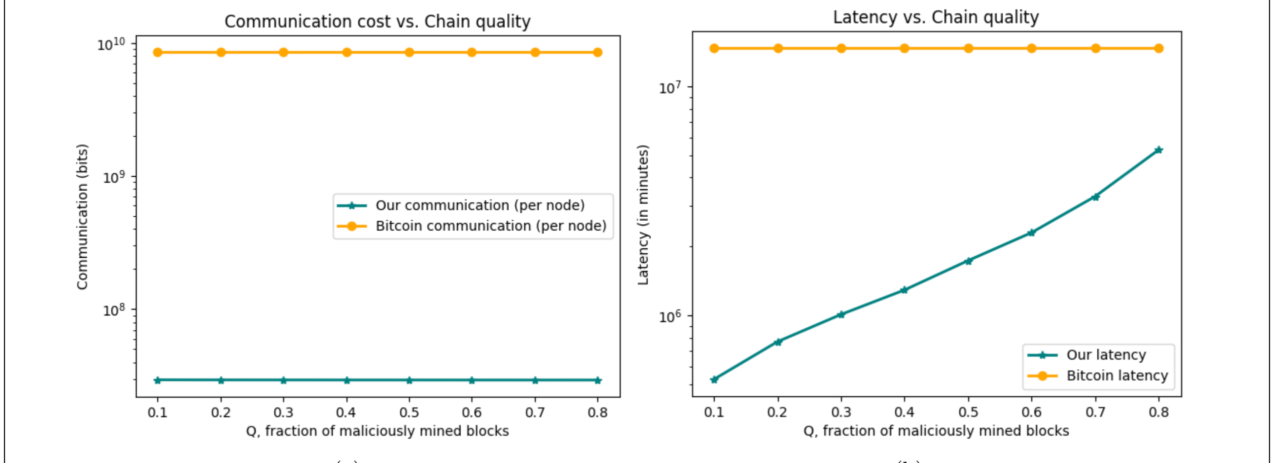


Figure 7: Analytical comparison of communication and latency when Q, the chain quality, changes.

6.1 Implementation and integration consideration

Implementation setup. We implemented our construction on top of Bitcoin (using Bitcoin core [5] as a black-box with extra lines of codes for our protocol). The implementation uses the Regression Test Network (regtest) mode to test our blob proposing, voting, and validating. There are a couple of considerations we kept in mind as developing the implementation:

1. We would like Bitcoin miners not to need additional computation for our protocol. Therefore our PoW puzzle is the same as in Bitcoin's (with distinct difficulties). This is to avoid attacks discussed in Section 5.2.1.

2. We need our data to be posted on the chain in a format compatible with the current Bitcoin standards. This is to ensure backward compatibility with the protocol.

For data availability sampling, we use the KZG commitment protocol [25] implemented in Go language [4]. The entire implementation is about 500 lines of code, working on top of Bitcoin, *without* the need to change the underlying Bitcoin protocol. The overall communication costs match the estimation above with some minor caveats. We delve into them and general implementation and considerations below.

Including a signature public key in PoW. One implementation caveat we had to work around is that the Bitcoin block header only allows for four bytes of nonce. As aforementioned, we would like to embed additional information in the header so that it also serves as a PoW for our votes(we need the proof of work to include the public key of a digital signature scheme, which is 32 bytes). This is not compatible with the current implementation. Therefore, one way to solve it is to extend the nonce field from 4 bytes to 36 bytes to maximize the efficiency of our construction. This also requires a block header to be extended from 80 bytes to 112 bytes.

Alternatively, we can include the public key in the data field of the block's coinbase transaction (the transaction used by the miner to pay itself for the block reward), which is then hashed to a Merkle root together with other transactions used to generate the PoW proof.[23] To use this idea, we also require including additional data in the vote for the other participants to verify the signature (the validator would need to include the public key together with its Merkle path to show that it is indeed the key used to generate the vote; otherwise, the signed votes cannot be verified correctly). The downside of this alternative approach is

[23]This is in fact already used by miners as additional nonce space, since 4 bytes of nonce is not nearly enough randomness to generate a PoW with the current difficulty level.

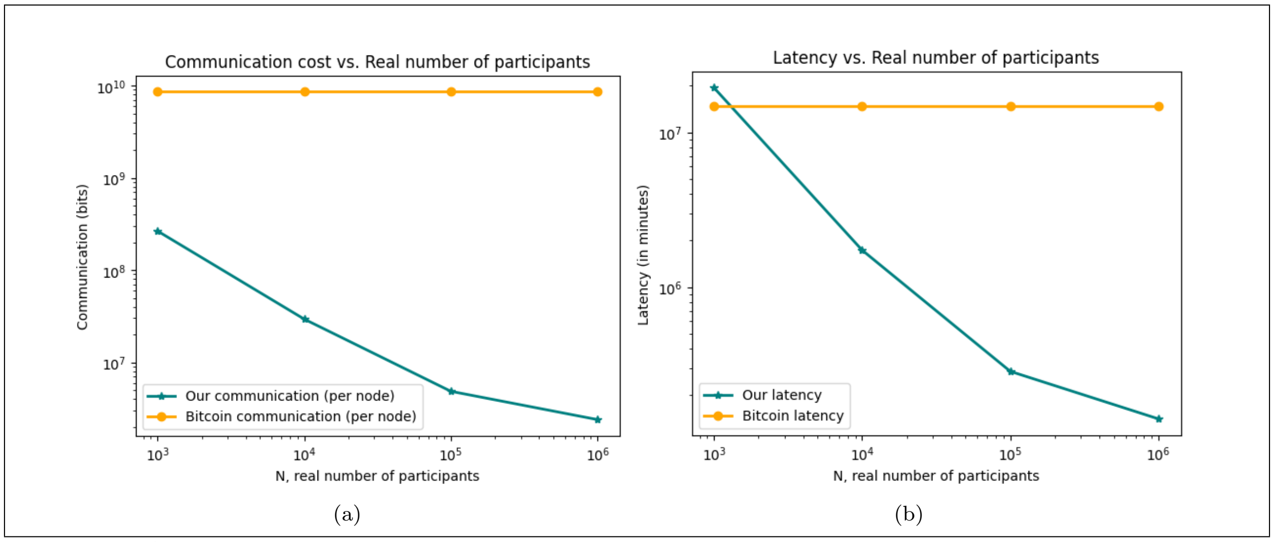


Figure 8: Analytical comparison of communication and latency when N, the real number of online participants, changes.

that it requires the Merkle path to be sent, which would not be needed if we had enough space to include the public key in the header (as mentioned above). Concretely, for a Merkle tree of 30 levels, each vote is increased by $30 \cdot 256$ bits. However, since the original vote is already about 6KB, this only adds an overhead of $< 15\%$ to the communication and computation. Thus, our advantages in communication and latency still similarly hold. Thus, one can either slightly extend the nonce field (and the block header) to maximize the efficiency, or simply attach a Merkle path, which thus does not require any change to the underlying Bitcore core implementation.

Posting votes Bitcoin. Another consideration is that, to include votes in a block in a way that's compatible with the current protocol, each vote needs to be made into a transaction, and the vote itself (i.e., the "yes" or "no" indicator for a particular commitment) needs to be included in the data field of a transaction. However, each transaction is about 215 bytes while the data field of it is only 80 bytes. Therefore, this incurs an overhead of 2.7x. Thus, to use Bitcoin as a black box, our communication cost is increased by 2.7x.Note that, however, this is similar to using Bitcoin directly. In other words, for the original Bitcoin protocol, to include β bits of data, there is also a 2.7x overhead if the current implementation is not changed to enlarge the data field accordingly.

We show in Appendix B that the integration overheads do not affect the results seen in Section 6 by much when looking at it in comparison to Bitcoin.

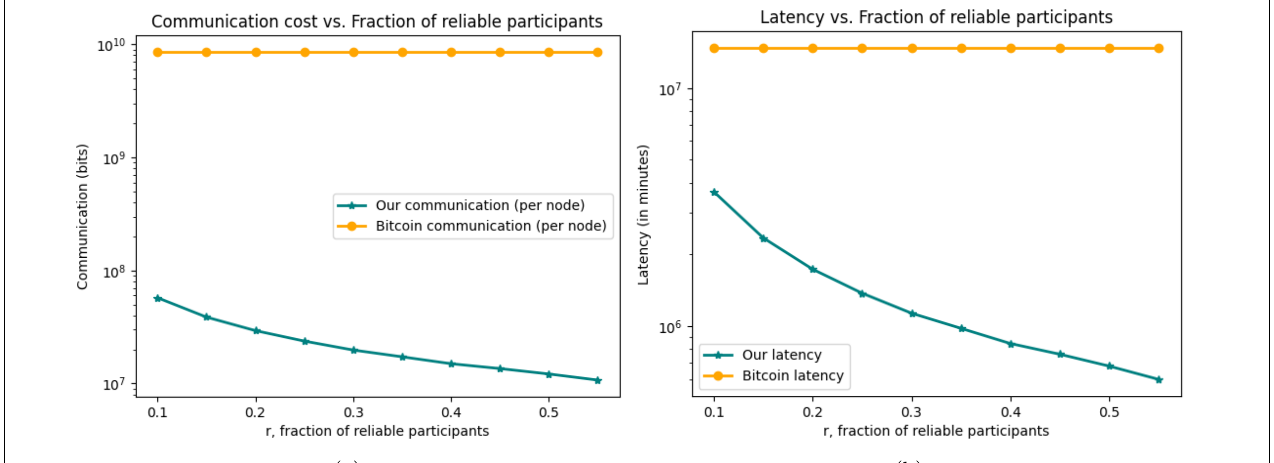


Figure 9: Analytical comparison of communication and latency when r, the fraction or reliable participants, changes.

References

[1] Celestia. `https://docs.celestia.org/learn/how-celestia-works/data-availability-layer`.

[2] Danksharding. `https://ethereum.org/en/roadmap/danksharding/`.

[3] Eigenda. `https://docs.eigenlayer.xyz/eigenda/overview/`.

[4] KZG Scheme in Go lang. `https://github.com/protolambda/go-kzg`, 2022.

[5] Bitcoin Core. `https://github.com/bitcoin/bitcoin`, 2024.

[6] M. Al-Bassam. Lazyledger: A distributed data availability ledger with client-side smart contracts. *arXiv preprint arXiv:1905.09274*, 2019.

[7] M. Al-Bassam, A. Sonnino, and V. Buterin. Fraud and data availability proofs: Maximising light client security and scaling blockchains with dishonest majorities. *arXiv preprint arXiv:1809.09044*, 2018.

[8] B. Bünz, J. Bootle, D. Boneh, A. Poelstra, P. Wuille, and G. Maxwell. Bulletproofs: Short proofs for confidential transactions and more. In *2018 IEEE Symposium on Security and Privacy*, pages 315–334, San Francisco, CA, USA, May 21–23, 2018. IEEE Computer Society Press.

[9] V. Buterin. An Incomplete Guide to Rollups, 2021.

[10] V. Buterin. Ethereum has blobs. Where do we go from here?, 2024.

[11] C. Cachin and S. Tessaro. Asynchronous verifiable information dispersal. In *24th IEEE Symposium on Reliable Distributed Systems (SRDS'05)*, pages 191–201, 2005.

[12] M. Campanelli, A. Nitulescu, C. Ràfols, A. Zacharakis, and A. Zapico. Linear-map vector commitments and their practical applications. In S. Agrawal and D. Lin, editors, *ASIACRYPT 2022, Part IV*, volume 13794 of *LNCS*, pages 189–219, Taipei, Taiwan, Dec. 5–9, 2022. Springer, Heidelberg, Germany.

[13] S. Celi, S. Griffy, L. Hanzlik, O. P. Kempner, and D. Slamanig. Sok: Signatures with randomizable keys. Cryptology ePrint Archive, Paper 2023/1524, 2023. `https://eprint.iacr.org/2023/1524`.

[14] S. Chaliasos, I. Reif, A. Torralba-Agell, J. Ernstberger, A. Kattis, and B. Livshits. Analyzing and benchmarking ZK-rollups. Cryptology ePrint Archive, Paper 2024/889, 2024. https://eprint.iacr.org/2024/889.

[15] A. Chator, M. Green, and P. R. Tiwari. Sok: Privacy-preserving signatures. Cryptology ePrint Archive, Paper 2023/1039, 2023. https://eprint.iacr.org/2023/1039.

[16] Coinwarz. Coinwarz Bitcoin Hashrate Chart. https://www.coinwarz.com/mining/bitcoin/hashrate-chart. Date: 04/16/2024, 2024.

[17] B. Core. Bitcoin core Source Code. https://github.com/bitcoin/bitcoin. Commit: 312f54278fd972ba3557c6a5b805fd244a063959. File: https://github.com/bitcoin/bitcoin/blob/master/src/pow.h, 2024.

[18] J. Garay, A. Kiayias, and N. Leonardos. The Bitcoin Backbone Protocol: Analysis and Applications. In E. Oswald and M. Fischlin, editors, *Advances in Cryptology - EUROCRYPT 2015*, pages 281–310, Berlin, Heidelberg, 2015. Springer.

[19] J. Garay, A. Kiayias, and N. Leonardos. How Does Nakamoto Set His Clock? Full Analysis of Nakamoto Consensus in Bounded Delay Networks. 2023.

[20] K. Gogol, Y. Velner, B. Kraner, and C. Tessone. Sok: Liquid staking tokens (lsts). *arXiv preprint arXiv:2404.00644*, 2024.

[21] M. Hall-Andersen, M. Simkin, and B. Wagner. Foundations of data availability sampling. Cryptology ePrint Archive, Paper 2023/1079, 2023. https://eprint.iacr.org/2023/1079.

[22] J. Hendricks, G. R. Ganger, and M. K. Reiter. Verifying distributed erasure-coded data. In I. Gupta and R. Wattenhofer, editors, *26th ACM PODC*, pages 139–146, Portland, OR, USA, Aug. 12–15, 2007. ACM.

[23] A. Jain and E. S. Pilli. Sok: Digital signatures and taproot transactions in bitcoin. In V. Muthukkumarasamy, S. D. Sudarsan, and R. K. Shyamasundar, editors, *Information Systems Security*, pages 360–379, Cham, 2023. Springer Nature Switzerland.

[24] H. A. Kalodner, S. Goldfeder, X. Chen, S. M. Weinberg, and E. W. Felten. Arbitrum: Scalable, private smart contracts. In W. Enck and A. P. Felt, editors, *USENIX Security 2018*, pages 1353–1370, Baltimore, MD, USA, Aug. 15–17, 2018. USENIX Association.

[25] A. Kate, G. M. Zaverucha, and I. Goldberg. Constant-size commitments to polynomials and their applications. In M. Abe, editor, *ASIACRYPT 2010*, volume 6477 of *LNCS*, pages 177–194, Singapore, Dec. 5–9, 2010. Springer, Heidelberg, Germany.

[26] P. McCorry, C. Buckland, B. Yee, and D. Song. SoK: Validating bridges as a scaling solution for blockchains. Cryptology ePrint Archive, Paper 2021/1589, 2021. https://eprint.iacr.org/2021/1589.

[27] S. Nakamoto. Bitcoin: A peer-to-peer electronic cash system. 2008.

[28] K. Nazirkhanova, J. Neu, and D. Tse. Information dispersal with provable retrievability for rollups. In *Proceedings of the 4th ACM Conference on Advances in Financial Technologies*, AFT '22, page 180–197, New York, NY, USA, 2023. Association for Computing Machinery.

[29] V. Nikolaenko and D. Boneh. Data availability sampling and danksharding: An overview and a proposal for improvements - a16z crypto, 2023.

[30] R. Pass and E. Shi. The sleepy model of consensus. In *Advances in Cryptology–ASIACRYPT 2017: 23rd International Conference on the Theory and Applications of Cryptology and Information Security, Hong Kong, China, December 3-7, 2017, Proceedings, Part II 23*, pages 380–409. Springer, 2017.

[31] Y. Pu, A. Farahbakhsh, L. Alvisi, and I. Eyal. Gorilla: Safe permissionless byzantine consensus. In *International Symposium on Distributed Computing*, 2023.

[32] L. Ren. Analysis of nakamoto consensus. Cryptology ePrint Archive, Paper 2019/943, 2019. https://eprint.iacr.org/2019/943.

[33] L. T. Thibault, T. Sarry, and A. S. Hafid. Blockchain scaling using rollups: A comprehensive survey. *IEEE Access*, 10:93039–93054, 2022.

[34] B. Whitehat. Roll-up token. https://github.com/barryWhiteHat/roll_up_token?tab=readme-ov-file#roll_up-token-snark-based-multi-erc20-side-chain, 2018.

[35] G. Wood. Ethereum: A secure decentralised generalised transaction ledger.

[36] L. Yang, S. J. Park, M. Alizadeh, S. Kannan, and D. Tse. DispersedLedger: High-Throughput byzantine consensus on variable bandwidth networks. In *19th USENIX Symposium on Networked Systems Design and Implementation (NSDI 22)*, pages 493–512, Renton, WA, Apr. 2022. USENIX Association.

[37] M. Yu, S. Sahraei, S. Li, S. Avestimehr, S. Kannan, and P. Viswanath. Coded merkle tree: Solving data availability attacks in blockchains. In J. Bonneau and N. Heninger, editors, *FC 2020*, volume 12059 of *LNCS*, pages 114–134, Kota Kinabalu, Malaysia, Feb. 10–14, 2020. Springer, Heidelberg, Germany.

A Additional Remarks

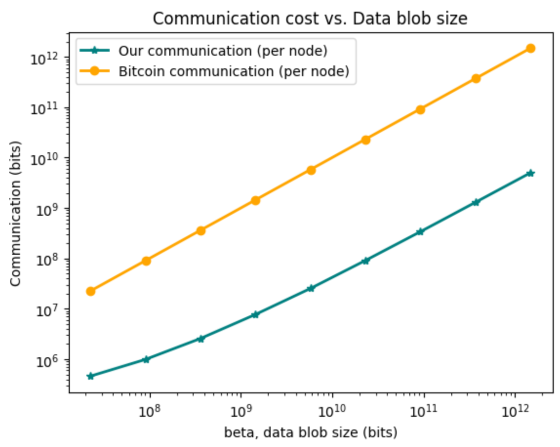
Remark A.1. *There are some existing works on permissionless BA, e.g. [31]. However, they are very inefficient: their communication can be exponential under some conditions. On the other hand, note that SMR indeed implies BA. Our construction, as mentioned in our main body, is also (implicitly) trying to achieve BA using SMR, and we need to achieve it efficiently under the permissionless setting.*

Remark A.2. *One thing to note is that the reliable participants can be different and independent for each data blob. In other words, a participant may be reliable only for a particular data blob (block) instead of always being reliable, and this is already sufficient for our construction. In particular, the participant only needs to reply to the reconstruction request to the particular blob(s) that it is reliable against. The only requirement we need is that for each data blob, there is r fraction of participants sampling it being reliable against this blob. This is a weaker assumption than having participants a set of participants that are always reliable throughout the entire blockchain.*

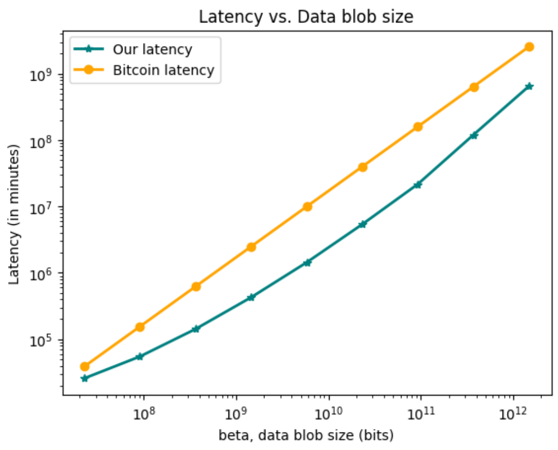
B Additional Evaluation

As mentioned in section 6.1, we count all the overhead we needed to implement atop Bitcoin without changing the underlying Bitcoin core implementation, and show our communication and latency costs compared to the current Bitcoin protocol.

As shown in figs. 10 to 16, our construction remains a similar advantage, except that the overhead makes the communication and latency about 3.1x worse compared to the optimal efficiency shown in section 6.

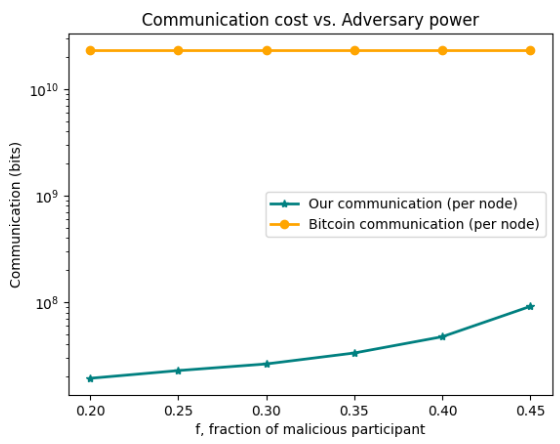


(a)

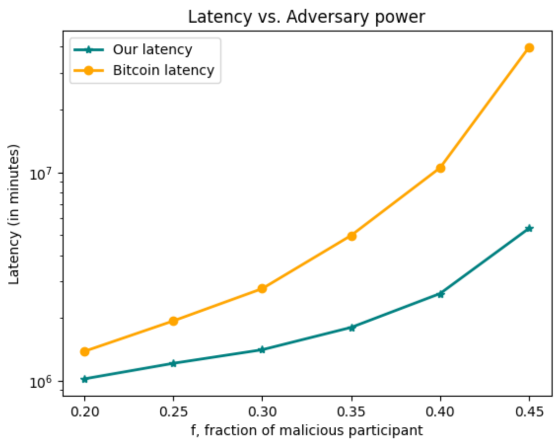


(b)

Figure 10: Implementation-based comparison of communication and latency when β, the blob size, changes. Integration overhead discussed in section 6.1 is considered.

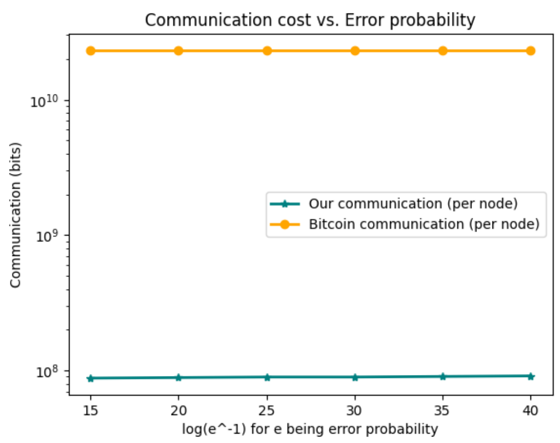


(a)

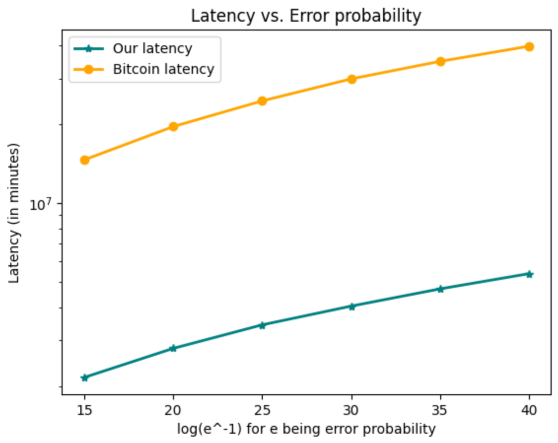


(b)

Figure 11: Implementation-based comparison of communication and latency when f, the fraction of malicious computation power, changes. Integration overhead discussed in section 6.1 is considered.

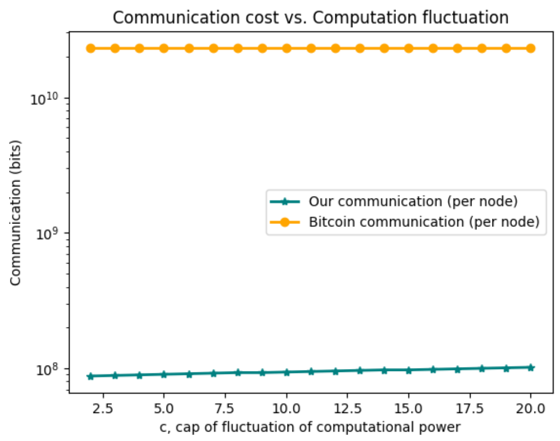


(a)

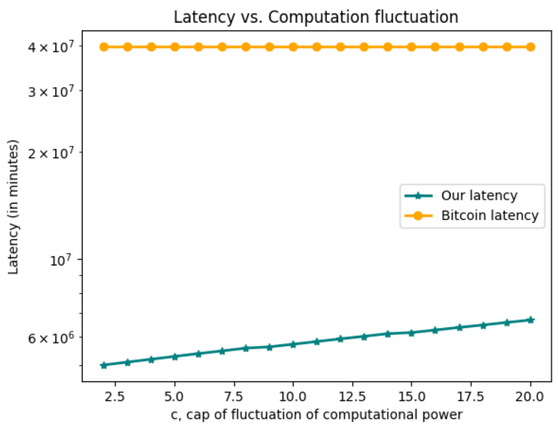


(b)

Figure 12: Implementation-based comparison of communication and latency when e, the error probability, changes. Integration overhead discussed in section 6.1 is considered.



(a)



(b)

Figure 13: Implementation-based comparison of communication and latency when c, the cap of computation power volatility, changes. Integration overhead discussed in section 6.1 is considered.

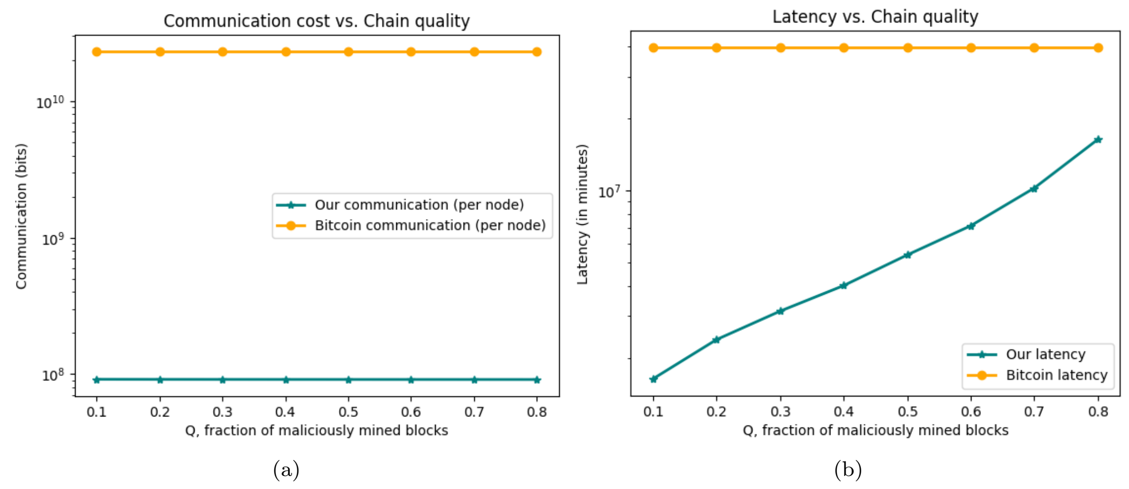


Figure 14: Implementation-based comparison of communication and latency when Q, the chain quality, changes. Integration overhead discussed in section 6.1 is considered.

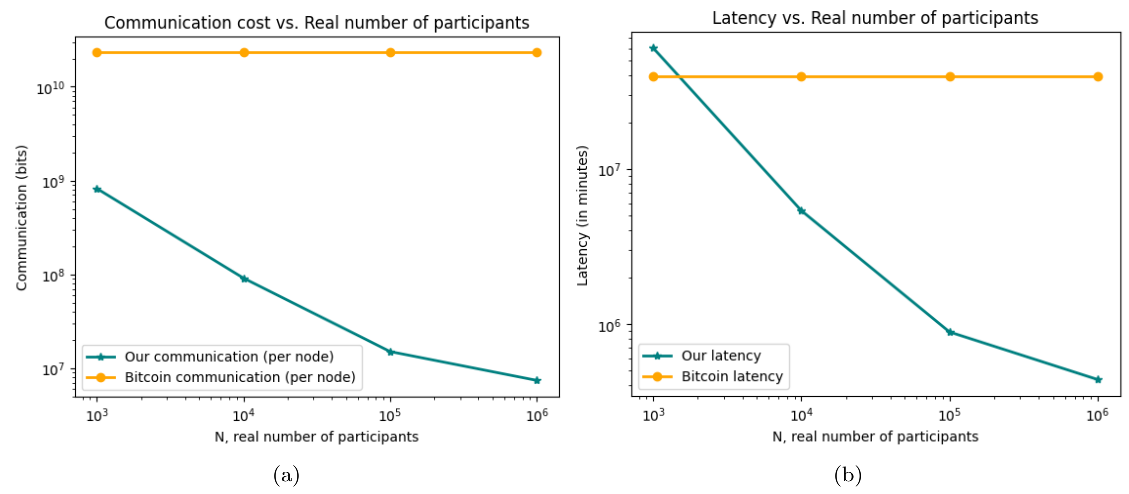
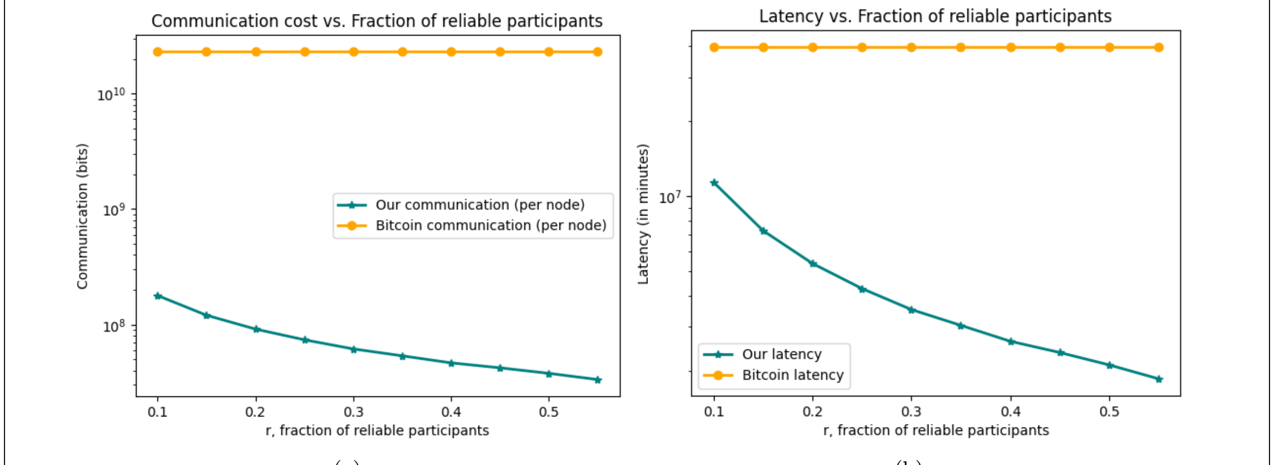


Figure 15: Implementation-based comparison of communication and latency when N, the real number of online participants, changes. Integration overhead discussed in section 6.1 is considered.



(a) (b)

Figure 16: Implementation-based comparison of communication and latency when r, the fraction or reliable participants, changes. Integration overhead discussed in section 6.1 is considered.