

Bandwidth-Balanced Reliable Broadcast in Stake-Weighted Networks

Victor Shoup

Category Labs
victor@shoup.net

October 4, 2026

Abstract. Consider an asynchronous network of n validators in which Byzantine validators hold less than one third of the stake. In erasure-coded reliable broadcast, validators relay fragments of the sender’s message. A validator’s **relay-upload cost** is the amount of fragment data it transmits outside the sender’s initial dispersal. Assigning each validator a number of fragments proportional to its stake makes the heaviest validators relay far more data than the rest. We show that this imbalance is unnecessary. We give **Balanced CT** and **Balanced MiniCast**, adaptations of Cachin–Tessaro (CT) and MiniCast (Locher and Shoup, EUROCRYPT 2025) that retain stake-weighted voting but assign one fragment to each validator, giving every honest validator the same relay-upload bound. Their maximum relay-upload costs depend on how few validators can exceed one third of the total stake for CT, or two thirds for MiniCast. Stake skew can amplify MiniCast’s advantage beyond the approximately twofold improvement in equal-stake networks. For the real-world Monad distribution with $n = 200$, the leading maximum relay-upload costs, as multiples of the message size, are approximately 72.2 for CT with proportional assignment, 8.7 for Balanced CT, and 2.2 for Balanced MiniCast.

The balanced constructions pay for their low maximum cost with a higher total relay-upload cost. To reduce that total while retaining a prescribed maximum-cost bound, we give a new **Capped Swiper** heuristic. On eight real-world stake distributions, with at most $4n$ fragments per code, it lowers total relay-upload cost by 19–81% for CT and 16–71% for MiniCast relative to their balanced counterparts, while keeping maximum relay-upload cost at most 10% higher. We also give sender-aware variants of both constructions in which the sender relays no fragment data, reducing its per-broadcast burden.

1 Introduction

Reliable broadcast (RBC) is a basic primitive in asynchronous Byzantine fault-tolerant protocols. Many communication-efficient RBC protocols use erasure coding [4,9]. Write $|M|$ for the bit length of a message M . An (N, K) **erasure code** encodes M into N **fragments**, each of size about $|M|/K$, so that any K fragments suffice to reconstruct M . We call K the **reconstruction threshold** of the code. The sender encodes M and disperses the fragments among the validators. The validators then relay their assigned fragment data to all validators. We call this basic structure the **disperse-and-relay paradigm**. A protocol in this paradigm may also use short control messages and a mechanism for recovering missing fragments.

We call the amount of fragment data transmitted by a validator outside the sender’s initial dispersal its **relay-upload cost**, and the sum over all validators the **total relay-upload cost**. Our primary bandwidth metric is the maximum relay-upload cost over validators. We focus on large messages: we report the coefficient of $|M|$ and treat terms independent of $|M|$ as lower order. Data that authenticates fragments, and short control messages, are accounted for separately, as is the **sender-upload cost**, the amount of fragment data transmitted by the sender in the initial dispersal.

In a stake-weighted validator network with validators $P_1, \dots, P_n$, let w_i be the stake of P_i and let $W := \sum_{i=1}^n w_i$ be the total stake. Let B be the Byzantine set, and write $w(B) := \sum_{i \in B} w_i$. We assume $w(B) < W/3$.

A standard way to adapt the erasure code to the weighted setting gives each physical validator a number of virtual validators proportional to its stake, with one fragment assigned to each virtual validator. We call this **proportional virtualization**. Swiper [15] gives a systematic way to keep the number of virtual validators small while preserving the relevant weighted threshold properties. Stake-proportional assignment is also used in practice. RaptorCast [7], Monad’s block dissemination protocol, follows the disperse-and-relay paradigm and sends each validator a number of fragments proportional to its stake. EigenDA [6] assigns erasure-coded chunks, and Walrus [5] storage shards, in proportion to stake. These systems solve different problems, but they raise the same question: how should erasure-coded data be assigned under stake-weighted fault assumptions? For reliable broadcast, the assignment directly determines how much fragment data each validator may have to relay. Let $p_i := w_i/W$ be the stake fraction of P_i , and write $p_{\max} := \max_i p_i$. We write CT for the erasure-coded RBC protocol of Cachin and Tessaro [4], and **Proportional CT** for CT with proportional virtualization. With V virtual validators in total, Proportional CT uses a (V, k) erasure code with $k \approx V/3$, and P_i relays approximately $p_i V$ fragments of size approximately $3|M|/V$, so its relay-upload cost is approximately $3np_i|M|$, and the maximum relay-upload cost is approximately $3np_{\max}|M|$.

Thus a validator’s relay-upload cost is proportional to its stake: for the Monad stake distribution used in our evaluation, $n = 200$, the heaviest validator holds about 12.0% of the total stake and the average validator holds 0.5%, so under proportional virtualization the heaviest validator relays roughly 24 times the average, and needs a correspondingly faster outbound connection. This imbalance may seem unavoidable: if a validator has 24 times the average stake, perhaps it must relay 24 times the average amount of fragment data. It need not. Stake determines voting power, but it does not determine how much fragment data each validator must relay. We call a construction **bandwidth-balanced** if its bound on a validator’s relay-upload cost does not depend on that validator’s stake; the bound may still depend on the stake distribution as a whole. We ask how to reduce the maximum relay-upload cost, what this costs in total relay traffic, and how the tradeoff depends on the protocol and the number of fragments.

The protocol logic determines which validator sets must collectively hold enough fragments to reconstruct. If a protocol requires reconstruction from every validator set of stake greater than θW , where $0 < \theta < 1$, we call θ the **stake threshold** of its code, and let ν_θ be the smallest number of validators whose combined stake exceeds θW , that is, the smallest k such that the k largest validators together hold more than θW .

In CT, a validator may have to reconstruct M from the fragments of the honest validators in a set of stake greater than $2W/3$, and these honest validators may hold only slightly more than $W/3$; that is, $\theta = 1/3$. In Section 4 we give **Balanced CT**, which uses an $(n, \nu_{1/3})$ erasure code and assigns one fragment to each validator. Its maximum relay-upload cost has leading term $(n/\nu_{1/3})|M|$, which matches a lower bound for the deterministic disperse-and-relay setting (Section 3), and is less than $3np_{\max}|M|$ for every stake distribution. For Monad, $\nu_{1/3} = 23$: Proportional CT gives $3np_{\max}|M| \approx 72.21|M|$, and Balanced CT gives $(n/\nu_{1/3})|M| \approx 8.70|M|$. Bammert et al. [2] give the same construction and the corresponding per-node fragment-size lower bound, but their objective is total fragment size (Section 8).

The choice of RBC protocol can further improve the tradeoff. MiniCast [10] follows the same disperse-and-relay paradigm and adds a recovery mechanism for missing fragments: with at most $t < n/3$ Byzantine parties, its main erasure code reconstructs from $n-t$ fragments, and its recovery code reconstructs a missing fragment from $n-2t$ smaller fragments. In our stake-weighted adaptation, **Balanced MiniCast**, the main code must reconstruct from validator sets of stake greater than $2W/3$ and the recovery code from sets of stake greater than $W/3$, so we use $k_1 := \nu_{2/3}$ and $k_2 := \nu_{1/3}$. The resulting leading relay-upload term for each validator is

$$\left(\frac{n}{\nu_{2/3}} + \frac{n}{\nu_{2/3}\nu_{1/3}} \right) |M|.$$

The lower bound for $\theta = 1/3$ does not apply here, because the main code has $\theta = 2/3$; the first term matches the lower bound for $\theta = 2/3$, and the second is the cost of recovery. Both Balanced CT and Balanced MiniCast are bandwidth-balanced. For the same Monad distribution, $\nu_{2/3} = 94$, and the leading relay-upload term is approximately $2.22|M|$, against approximately $(3n/2 + 9/2)p_{\max}|M| \approx 37|M|$ for **Proportional MiniCast**, MiniCast with proportional virtualization. Stake skew amplifies the benefit: with equal stakes, $\nu_{2/3} \approx 2\nu_{1/3}$, but for Monad, $\nu_{2/3} \approx 4\nu_{1/3}$, because the first third of the stake is held by few validators. Thus the same skew that makes proportional virtualization unbalanced makes a main code with reconstruction threshold $\nu_{2/3}$ more effective.

These reductions in maximum relay-upload cost can come at the cost of additional total relay traffic. With one fragment of an (n, ν_θ) erasure code per validator, the leading total relay-upload cost is $(n^2/\nu_\theta)|M|$, against approximately $(n/\theta)|M|$ for proportional virtualization at the same stake threshold, a ratio of approximately $\theta n/\nu_\theta$: about 2.90 for Monad at $\theta = 1/3$. Balanced MiniCast has substantially lower total relay-upload cost than Balanced CT, because its main code has the larger reconstruction threshold $\nu_{2/3}$; for Monad, its total is about 1.5 times that of Proportional MiniCast.

The total can be lowered while keeping each code’s maximum relay-upload cost near its minimum. In [Section 7](#) we assign different numbers of fragments to different validators, and find such fragment assignments with **Capped Swiper**, an adaptation of Swiper [15]. With that maximum at most 10% above its minimum for each code and at most $4n$ fragments in each code, this lowers the average relay-upload cost by between 16% and 71% for MiniCast and between 19% and 81% for CT relative to the balanced protocols on eight real-world stake distributions ([Section 6](#)), while the maximum relay-upload cost of either protocol is at most 10% above that of its balanced counterpart.

The rest of the paper. [Section 2](#) fixes the network and adversary model and defines reliable broadcast. [Section 3](#) recalls CT, gives its stake-weighted voting logic, and proves the relay-upload lower bound. [Section 4](#) gives Balanced CT. [Section 5](#) introduces and analyzes Balanced MiniCast, its direct-repair variant, and sender-aware variants of Balanced CT and Balanced MiniCast. [Section 6](#) compares the constructions with proportional virtualization and Swiper on eight real-world validator stake distributions. [Section 7](#) introduces Capped Swiper for finding fragment assignments that trade total relay-upload cost against maximum relay-upload cost. We discuss related work in [Section 8](#) and conclude in [Section 9](#).

2 Model and notation

There are parties $P_1, \dots, P_n$, which we also call validators, with nonnegative weights $w_1, \dots, w_n$ and total weight $W = \sum_i w_i > 0$. The weight w_i is the weight that the consensus protocol assigns to P_i ; we call it the stake of P_i . Write $[n] := \{1, \dots, n\}$. For $S \subseteq [n]$, write

$$w(S) := \sum_{i \in S} w_i.$$

The adversary statically corrupts a set B satisfying $w(B) < W/3$. Communication is asynchronous over authenticated point-to-point channels. The adversary schedules message delivery and controls Byzantine parties, but cannot alter messages sent between honest parties. We use the formulation of reliable broadcast of [10,14], whose two properties are direct analogues of the corresponding properties of verifiable secret sharing.

Definition 2.1 (Reliable broadcast). *A **reliable broadcast** protocol allows a designated sender P_s to broadcast a single message to all parties. It must satisfy the following properties.*

Correctness. (i) *Any two honest parties that output a message output the same message.* (ii) *If the sender is honest, any honest party that outputs a message outputs the sender’s input message.*

Completeness. *If some honest party has output a message, or the sender is honest and has received an input, then all honest parties eventually output a message.*

When cryptographic primitives are used, these properties are required to hold except with negligible probability against probabilistic polynomial-time adversaries under the relevant cryptographic assumptions.

As in [10], “eventually” means if and when all messages already sent between honest parties have been delivered.

3 Disperse-and-relay setting

Classical erasure-coded RBC. To fix ideas, first consider an unweighted system with $n = 3t + 1$ parties and at most t Byzantine parties. We recall CT [4], in the simplified presentation of Shoup and Smart [14, Fig. 5]. The sender encodes M with an $(n, t + 1)$ erasure code. It commits to the n fragments with a Merkle tree with root r , and sends each party r and one fragment together with its **validation path** under r ; we call the pair a **certified fragment** for r if the path is correct. A party acts only on the first message it receives from the sender: if that message contains a certified fragment for a root r , the party **echoes** r and the certified fragment to all parties. This is the fragment relay of the disperse-and-relay paradigm. A party that has not yet voted votes for a root after receiving, for that root, either echoes from $2t + 1$ parties or votes from $t + 1$ parties (the latter is Bracha’s amplification step [3]). After receiving votes for a root from $2t + 1$ parties, a party waits until it has $t + 1$ certified fragments for that root. It then reconstructs the message, re-encodes it, and outputs it if the resulting Merkle root is the voted root; otherwise, it outputs $\perp$.

Weighted voting logic. We call the condition that enables a protocol step its **guard**. In a stake-weighted network, the cardinality guards in the unweighted voting logic can be replaced by exact stake guards on the physical validators. A party that has not yet voted votes for a root after receiving, for that root, either echoes from validators of stake greater than $2W/3$ or votes from validators of stake greater than $W/3$. After receiving votes for a root from validators of stake greater than $2W/3$, a party waits until it has enough certified fragments for that root to reconstruct.

An honest party that votes after receiving echoes has received them from validators of stake greater than $2W/3$. The honest validators among them have stake greater than $W/3$, and they echoed certified fragments to all parties; a party that is waiting to reconstruct may receive no other fragments. The erasure code must therefore ensure that every set of physical validators of stake greater than $W/3$ collectively holds enough fragments to reconstruct the message.

Proportional CT. In Proportional CT, each physical validator P_i is replaced by an integer number v_i of virtual validators. We call $v = (v_1, \dots, v_n)$ a **virtualization**. Each of the $V := \sum_i v_i$ virtual validators is assigned one fragment of a (V, k) erasure code, and the physical validator acts on behalf of all of its virtual validators. The requirement becomes that every set of physical validators of stake greater than $W/3$ controls at least k virtual validators. This is Swiper’s weight qualification condition [15]; Swiper finds small v_i that satisfy it among virtualizations proportional to stake up to rounding. The voting logic still uses the physical validators directly; the virtual validators are used only to allocate fragments. Because a physical validator relays the fragments assigned to all of its virtual validators, the amount of fragment data it relays grows with its stake. We quantify this in Section 6.1.

General abstraction. Other protocols can induce different stake thresholds, so we now state the coding problem more generally. We consider deterministic dispersal. The sender deterministically assigns fragment data $X_i(M)$ to each validator P_i , where $X_i(M)$ may contain zero, one, or several coded fragments. Each validator then relays all fragment data it holds to all validators. Relay-upload cost and sender-upload cost are as defined in Section 1, measured in bits. For simplicity, sending the same fragment data to all validators counts as n transmissions. For each message length ℓ and validator P_i , assume that $X_i(M)$ has a fixed finite bit length $L_i(\ell)$, independent of the particular message M among messages with $|M| = \ell$. When a message length ℓ is fixed, we write L_i for $L_i(\ell)$. The protocol logic determines which validator sets must collectively hold enough fragment data to reconstruct. The following lower bound applies to the deterministic disperse-and-relay setting above, and to protocols that add messages to it, such as MiniCast’s recovery mechanism (Section 5), since added messages only raise relay-upload cost; it does not apply to arbitrary reliable broadcast protocols.

Theorem 3.1 (Relay-upload lower bound). *Fix $0 < \theta < 1$ and suppose that reconstruction is required from every validator set $S \subseteq [n]$ with $w(S) > \theta W$. Define*

$$\nu_\theta := \min\{|S| : S \subseteq [n] \text{ and } w(S) > \theta W\}.$$

Equivalently, ν_θ is the length of the shortest descending-by-stake prefix whose total stake exceeds θW . Then every construction in the setting above has maximum relay-upload cost at least

$$\frac{n}{\nu_\theta} |M|.$$

Proof. Fix the message length $\ell = |M|$. Let S^* be a minimum-cardinality set with $w(S^*) > \theta W$, so $|S^*| = \nu_\theta$. The reconstruction requirement implies that the map

$$M \mapsto (X_i(M))_{i \in S^*}$$

is injective on the 2^ℓ possible messages. The tuples have at most $2^{\sum_{i \in S^*} L_i}$ possible values, so

$$2^{\sum_{i \in S^*} L_i} \geq 2^\ell.$$

Hence

$$\sum_{i \in S^*} L_i \geq \ell,$$

and therefore $L_i \geq \ell/\nu_\theta$ for some $i \in S^*$. That validator relays its fragment data to all n validators, giving relay-upload cost at least $n\ell/\nu_\theta$. $\square$

The corresponding per-node fragment-size lower bound is observed in [2, Section 6.2].

4 Balanced CT

We now give **Balanced CT**, whose relay-upload cost matches the lower bound of [Theorem 3.1](#) for $\theta = 1/3$ in the leading term. It is CT, as recalled in [Section 3](#), with two changes. First, it uses the weighted voting logic of [Section 3](#). Second, the sender encodes M with an $(n, \nu_{1/3})$ erasure code, so a party reconstructs from $\nu_{1/3}$ certified fragments. Each validator still holds one fragment. This meets the requirement of [Section 3](#) without virtualization: by definition of $\nu_{1/3}$, every set of stake greater than $W/3$ contains at least $\nu_{1/3}$ validators. This is the code used in the proof of [2, Theorem 7], for the family of validator sets of stake greater than $W/3$.

Theorem 4.1 (Balanced CT). *Assume $w(B) < W/3$ and collision resistance of the underlying hash function. Balanced CT satisfies reliable broadcast, except with negligible probability, and each honest validator has relay-upload cost at most*

$$\frac{n}{\nu_{1/3}} |M| + O(n \log n).$$

Proof. The proofs of the Bracha properties B0–B3 in [14] use only quorum arguments, and go through with stake in place of cardinality. Reliable broadcast then follows as in [14], with one new step. By B2, if an honest party votes for a root r , then honest validators of total stake greater than $W/3$ echoed r , each with a certified fragment. By definition of $\nu_{1/3}$, there are at least $\nu_{1/3}$ of them, so every honest party eventually receives enough certified fragments to reconstruct. For the cost, an honest validator echoes at most once, sending one fragment to each of the n validators. With a standard Reed–Solomon realization, a fragment has size $|M|/\nu_{1/3} + O(\log n)$ bits. $\square$

Validation paths and short control messages are accounted for separately. The coefficient $n/\nu_{1/3}$ is also always smaller than the coefficient $3np_{\max}$ of Proportional CT, as the following proposition shows.

Proposition 4.2. *For every stake distribution and every $0 < \theta < 1$,*

$$\frac{n}{\nu_\theta} < \frac{np_{\max}}{\theta}.$$

In particular, $n/\nu_{1/3} < 3np_{\max}$.

Proof. A set of ν_θ validators of stake greater than θW has stake at most $\nu_\theta p_{\max} W$. Hence $\nu_\theta p_{\max} > \theta$. $\square$

5 Balanced MiniCast

MiniCast [10] is an unweighted reliable broadcast protocol in which at most $t < n/3$ parties are Byzantine. Its main code reconstructs from $n - t$ fragments, and its recovery code reconstructs a missing fragment from $n - 2t$ smaller fragments. In **Balanced MiniCast**, the main code must reconstruct from validator sets of stake greater than $2W/3$, and the recovery code from sets of stake greater than $W/3$, so we set $k_1 := \nu_{2/3}$ and $k_2 := \nu_{1/3}$.

5.1 Data encoding

The data encoding is that of MiniCast, with reconstruction thresholds k_1 and k_2 in place of $n - t$ and $n - 2t$. The **main code** is an (n, k_1) erasure code. It encodes M as fragments $f_1, \dots, f_n$, any k_1 of which reconstruct M . The **recovery code** is an (n, k_2) erasure code. For each $i \in [n]$, it encodes f_i as **mini-fragments** $\phi_{i1}, \dots, \phi_{in}$, any k_2 of which reconstruct f_i . As we will see, party P_j sends ϕ_{ij} to P_i , so that P_i can recover f_i if it did not obtain it from the sender.

We use a two-level Merkle tree. For each $i \in [n]$, let r_i be the root of a Merkle tree whose leaves are the hashes of $\phi_{i1}, \dots, \phi_{in}$, and let r be the root of a Merkle tree whose leaves are $r_1, \dots, r_n$. We thus have validation paths π_i for r_i under r at position i , and validation paths π_{ij} for ϕ_{ij} under r_i at position j . The **tag** of M is $\tau := (|M|, r)$.

For a tag $\tau = (\ell, r)$, we call (f_i, π_i) a **certified fragment for τ at position i** if f_i has the correct length of a fragment for a message of length ℓ and, when we compute the mini-fragments of f_i and the root r_i of their Merkle tree, π_i is a correct validation path for r_i under r at position i . We call $(\phi_{ij}, \pi_{ij}, \pi_i)$ a **certified mini-fragment for τ at position (i, j)** if ϕ_{ij} has the correct length of a mini-fragment for a message of length ℓ , π_{ij} is a validation path for ϕ_{ij} under a root r_i at position j , and π_i is a correct validation path for r_i under r at position i .

The function *Encode* takes a message M and returns its tag τ together with a certified fragment (f_i, π_i) for τ at each position $i \in [n]$. The function *Decode*, called by P_i , takes a tag τ and certified fragments for τ at k_1 or more positions. It reconstructs a candidate message M' from k_1 of these fragments, re-encodes M' , and fails unless the resulting Merkle root matches the tag. If it succeeds, it returns M' together with a certified mini-fragment $(\phi'_{ji}, \pi'_{ji}, \pi'_j)$ for τ at each position (j, i) , $j \in [n]$. Consequently, except with negligible probability, any two successful calls to *Decode* on certified fragments under the same tag return the same message. The function *Recover*, called by P_i , takes certified mini-fragments for τ at k_2 or more positions (i, j) , and reconstructs the fragment f_i from k_2 of them. Collision resistance also implies, except with negligible probability, that certified data under the same tag and at the same position are unique.

5.2 Protocol

The protocol $\Pi_{\text{BMiniCast}}$ appears in Fig. 1 in Appendix A. It is obtained from MiniCast by replacing cardinality guards by stake guards. For a set $\mathcal{J}$ of distinct parties, the guard $|\mathcal{J}| = n - t$ becomes $w(\mathcal{J}) > 2W/3$, and the recovery guard $|\mathcal{J}| = n - 2t$ becomes $w(\mathcal{J}) > W/3$. The quantities k_1 and k_2 appear only as code parameters. By definition of ν_θ , a set of stake greater than $2W/3$ contains at least k_1 parties, and a set of stake greater than $W/3$ contains at least k_2 parties, so whenever a guard holds, the corresponding decoder has enough inputs. All guards count each party at most once, and count only messages that pass the required syntactic and cryptographic checks.

At a high level, the protocol proceeds in rounds, as in MiniCast.

1. In the *disperse* round, the sender computes $Encode(M)$ and sends the tag and one certified fragment to each party.
2. In the *echo* round, each party echoes the tag it received from the sender to all parties, provided that it came with a certified fragment.
3. In the *vote* round, each party waits for echo messages from stake greater than $2W/3$ with tags that match the tag it received from the sender; when this happens, it sends a vote message containing the tag and its certified fragment to all parties.
4. In the *confirm* round, each party P_i waits for vote messages from stake greater than $2W/3$ with matching tags; when this happens, it calls *Decode*. If *Decode* succeeds, P_i sets its candidate output M^* , which is initially $\perp$, and sends to each party P_j a confirm message containing the tag and the certified mini-fragment at position (j, i) .
5. A party outputs M^* once $M^* \neq \perp$ and it has received confirm messages from stake greater than $2W/3$.

We say that a party that votes as in the vote round votes *by echo*. In addition, there is a feedback loop. A party P_i that has not yet voted, for example because it did not receive a certified fragment from the sender, also votes after receiving confirm messages for the same tag from stake greater than $W/3$ that contain certified mini-fragments at positions (i, j) . It uses *Recover* to obtain its fragment f_i , which it sends in its vote message. We say that such a party votes *by confirmation*. Note that, in contrast to Balanced CT, echo messages here are short, and vote messages carry the fragments.

Fig. 1 omits one optimization from MiniCast, which we assume is always applied: a confirming party omits the mini-fragment from its confirm message to any party from which it has already received a vote.

5.3 Security

Theorem 5.1 (Security of Balanced MiniCast). *Assume $w(B) < W/3$ and collision resistance of the underlying hash function. Protocol $\Pi_{\text{BMiniCast}}$ satisfies the correctness and completeness properties of Definition 2.1, except with negligible probability.*

Proof. The proof of [10, Theorem 4.1] uses the thresholds $n - t$ and $n - 2t$ in two ways: in quorum arguments, which go through with stake in place of cardinality; and to ensure that *Decode* and *Recover* have enough inputs, which holds here by Section 5.2. For completeness, Appendix B gives the full proof. $\square$

5.4 Relay-upload cost

Theorem 5.2 (Balanced MiniCast relay-upload cost). *Let κ denote the hash-output length. Suppose that a fragment has size at most $|M|/k_1 + O(\log n)$ and a mini-fragment has size at most $|M|/(k_1 k_2) + O(\log n)$, where the additive terms account for padding to whole erasure-code symbols. Then each honest validator has relay-upload cost at most*

$$\left(\frac{n}{k_1} + \frac{n}{k_1 k_2} \right) |M| + O(n \log n),$$

and its total upload outside the sender's initial dispersal, including validation paths and control messages, is at most

$$\left(\frac{n}{k_1} + \frac{n}{k_1 k_2} \right) |M| + O(\kappa n \log n).$$

Proof. An honest validator sends at most one vote message containing one certified fragment to each of the n parties. The fragment data in these vote messages is therefore at most

$$n \left(\frac{|M|}{k_1} + O(\log n) \right).$$

It also sends at most one confirm message to each of the n parties. Each such confirm message contains one certified mini-fragment, whose size is at most $|M|/(k_1 k_2) + O(\log n)$. The fragment data in these confirm messages is therefore at most

$$n \left(\frac{|M|}{k_1 k_2} + O(\log n) \right).$$

Each certified fragment carries one validation path of size $O(\kappa \log n)$, and each certified mini-fragment carries two validation paths of total size $O(\kappa \log n)$. Echo messages and tags carry only lower-order data. Adding these terms gives the stated relay-upload cost and total-upload bound. $\square$

The proof ignores the optimization of [Section 5.2](#). At confirmation time a party has received votes from at least k_1 validators, so the optimization lowers the coefficient of $|M|$ in [Theorem 5.2](#) to

$$\frac{n}{k_1} + \frac{n - k_1}{k_1 k_2},$$

a saving of only $|M|/k_2$.

By [Proposition 4.2](#), the main term n/k_1 is less than $(3/2)np_{\max}$, the leading coefficient of Proportional MiniCast ([Section 6.1](#)). Since $k_2 \geq 1$, the coefficient of $|M|$ in [Theorem 5.2](#) is therefore less than $3np_{\max}$ for every stake distribution.

5.5 Direct-repair variant

A simpler variant keeps Balanced MiniCast’s main code and control flow but replaces the recovery code by direct repair of missing fragments. The variant can be viewed as Balanced MiniCast with a recovery code of reconstruction threshold 1, in which every mini-fragment of f_i is a copy of f_i . The proof of [Theorem 5.1](#) applies with $k_2 = 1$. The optimization of [Section 5.2](#) is essential here: without it, every confirming party would send a fragment to every validator. The variant avoids computing and hashing the n^2 mini-fragments.

Direct repair is the repair mechanism of Locher’s protocol [9]. As we show below, in the unweighted setting the variant matches the communication complexity of that protocol in the leading term. Locher’s protocol uses somewhat different logic, which gives it better round complexity: its good-case round complexity is 3, while that of MiniCast is 4 [10]. We do not reproduce that logic here. Simplicity, and with Locher’s logic better round complexity, are reasons to prefer direct repair when its higher relay-upload cost is acceptable.

After a party successfully decodes under a tag τ , it sends a short confirm message for τ to every validator. For a validator P_j from which it has not received a vote, the confirm also contains the certified fragment (f_j, π_j) . A validator that has not yet voted waits for confirm messages for the same tag from stake greater than $W/3$. One valid certified copy of its own fragment is sufficient for recovery, after which it can vote.

Without repair, a validator relays one fragment to all validators, giving leading relay-upload term

$$\frac{n}{\nu_{2/3}}|M|.$$

At confirmation time it has received votes from at least k_1 distinct validators, so it sends repair fragments to at most $n - k_1$ validators. The worst-case leading relay-upload term including repair is therefore

$$\left(2\frac{n}{\nu_{2/3}} - 1\right)|M|,$$

which is the optimized coefficient of [Section 5.4](#) with $k_2 = 1$. With equal stakes and $n = 3t + 1$, we have $\nu_{2/3} = n - t$, and this is approximately $2|M|$ per validator, matching the communication complexity of approximately $2|M|n$ of Locher’s protocol. The number of validators that need repair can be large. In the unweighted setting, at most $t < n/3$ parties are missing from a set of $n - t$ voters. With skewed stake, a set of voters of stake greater than $2W/3$ may have as few as $\nu_{2/3}$ members, so up to $n - \nu_{2/3}$ validators may be missing: 106 of 200 for the Monad distribution. Repair traffic is also not spread evenly. A lagging validator may receive up to one full set of vote fragments and, in the worst case, duplicate copies of its own fragment from all confirmers. The fragment data it receives is then at most

$$2\frac{n}{\nu_{2/3}}|M|,$$

twice that of a validator that does not lag, and the adversary’s scheduling determines which validators lag. The optimization has the same effect in Balanced MiniCast, but there a lagging validator receives at most $n|M|/(k_1k_2)$ more than the others.

5.6 Sender-upload cost and sender-aware variants

The bound in [Theorem 5.2](#) excludes the sender’s initial dispersal. We first quantify the sender-upload cost, and then give sender-aware variants of Balanced CT and Balanced MiniCast.

The relay-upload bound above applies to the sender in its receiving-party role as well as to every non-sender. Thus every honest validator has the same relay-upload bound, independent of its stake. The sender additionally sends one certified fragment to each party during the initial dispersal. Its sender-upload cost therefore has leading term

$$\frac{n}{k_1}|M|.$$

Combining this with the relay-upload bound gives the sender total fragment-data upload

$$\left(2\frac{n}{k_1} + \frac{n}{k_1k_2}\right)|M|$$

in the leading term.

Symmetric applications. In a **symmetric application**, every validator acts as sender once per batch. This pattern occurs in constructions of **asynchronous common subset (ACS)**, where each validator reliably broadcasts an input and the validators agree on a common subset of these inputs [12]. For equal-sized inputs, an otherwise symmetric unweighted RBC protocol has the same total-upload bound for every validator over the batch, even if the sender's cost is much larger than that of the other validators: each validator incurs the extra sender cost once. In proportional RBC, however, the stake-dependent relay burden persists in every instance, so taking turns as sender does not remove that imbalance. Balanced CT and Balanced MiniCast remove it without requiring sender-aware variants. For Balanced MiniCast, let

$$A := \frac{n}{k_1} + \frac{n}{k_1 k_2}.$$

In a batch of n equal-sized instances in which every validator is the sender once, each honest validator's average fragment-data upload per instance is at most

$$\left(A + \frac{1}{k_1}\right) |M|$$

in the leading term, so dispersal adds only a $1/n$ fraction to the dominant n/k_1 relay term.

Asymmetric applications. In an **asymmetric application**, one designated sender broadcasts a proposal, as in a leader-based consensus protocol [13]. The sender may rotate between proposals, but averaging over a rotation does not remove its bandwidth bottleneck within an individual broadcast. In one instance of Balanced MiniCast, the sender uploads up to twice as much as any other validator. Sender-aware variants can therefore be most useful in this setting: they eliminate the current sender's relay-upload cost, although the change in code parameters can increase other costs.

Sender-aware variants. MiniCast [10, Section 4.1.2], adapting an idea from [13], removes the sender's extra cost in the unweighted setting by using codes with $n - 1$ fragments, so that the sender holds no fragment and no mini-fragment. We apply this technique to Balanced CT and Balanced MiniCast. Excluding the sender eliminates its relay-upload cost, but it can change the reconstruction thresholds and hence the fragment sizes. Fix the sender P_s and write w_s for its stake. For $\theta \in (0, 1)$, let $\nu_{\theta,s}$ be the smallest number of non-senders that, together with P_s , have stake greater than θW :

$$\nu_{\theta,s} := \min\{|R| : R \subseteq [n] \setminus \{s\}, w_s + w(R) > \theta W\}.$$

Lemma 5.3. *For every $\theta \in (0, 1)$ and sender P_s ,*

$$\nu_{\theta} - 1 \leq \nu_{\theta,s} \leq \nu_{\theta},$$

and every $Q \subseteq [n]$ with $w(Q) > \theta W$ contains at least $\nu_{\theta,s}$ non-senders.

Proof. If R is feasible in the definition of $\nu_{\theta,s}$, then $R \cup \{s\}$ has stake greater than θW . Thus $|R| + 1 \geq \nu_{\theta}$, proving the lower bound. Now take a qualifying set of cardinality ν_{θ} . Removing s if it is present gives a feasible non-sender set of size $\nu_{\theta} - 1$; if it is absent, the set itself is feasible and has size ν_{θ} . Finally, if $w(Q) > \theta W$, then

$$w_s + w(Q \setminus \{s\}) \geq w(Q) > \theta W,$$

so $Q \setminus \{s\}$ is feasible and $|Q \setminus \{s\}| \geq \nu_{\theta,s}$. □

Sender-aware CT. Suppose $\nu_{1/3,s} > 0$. Encode M with an $(n - 1, \nu_{1/3,s})$ erasure code, assigning one fragment to each non-sender. The sender's absent fragment is represented by an empty value NULL, which is treated as certified only at the sender's position. An honest sender sends itself a dispersal message with this value and echoes the root like any other validator. All validators retain the weighted voting guards of Balanced CT, counting the sender's stake, but ignore NULL when reconstructing and use the Merkle tree on the $n - 1$ non-sender fragments. By Lemma 5.3, the honest echoing validators of stake greater than $W/3$ include at least $\nu_{1/3,s}$ non-senders, so the proof of Theorem 4.1 applies. In the leading term, the sender-upload cost is at most $((n - 1)/\nu_{1/3,s})|M|$, the sender's relay-upload cost is zero, and each honest non-sender has relay-upload cost at most $(n/\nu_{1/3,s})|M|$. If $\nu_{1/3,s} = 0$, equivalently $w_s > W/3$, use Balanced CT unchanged.

Sender-aware MiniCast. Set

$$k_{1,s} := \nu_{2/3,s}, \quad k_{2,s} := \nu_{1/3,s}.$$

We have $k_{2,s} = 0$ exactly when $w_s > W/3$. In that case the variant does not apply, and we use Balanced MiniCast unchanged. When $k_{2,s} > 0$, encode the message with an $(n - 1, k_{1,s})$ main code, assigning one fragment to each non-sender. Encode each fragment with an $(n - 1, k_{2,s})$ recovery code, assigning one mini-fragment to each non-sender. The sender holds no fragment and no mini-fragment. The NULL convention now also applies to every mini-fragment position that involves the sender, and only there. Otherwise the sender runs the protocol like any other party; in particular, it sends itself a dispersal message. *Decode* and *Recover* ignore NULL values, and *Recover* returns NULL at the sender's position.

Corollary 5.4 (Sender-aware MiniCast). *If $k_{2,s} > 0$, the sender-aware MiniCast construction above satisfies reliable broadcast under $w(B) < W/3$, assuming collision resistance of the underlying hash function.*

Proof. The construction runs Balanced MiniCast as if the sender's fragment and mini-fragments were NULL. The proof of Theorem 5.1 applies, once *Decode* and *Recover* have enough non-NULL inputs. By Lemma 5.3, every vote set of stake greater than $2W/3$ contains at least $k_{1,s}$ non-sender fragments, and every confirm set of stake greater than $W/3$ contains at least $k_{2,s}$ non-sender mini-fragments. $\square$

In the leading term, the sender-upload cost of the variant is at most

$$\frac{n - 1}{k_{1,s}}|M|.$$

Each honest non-sender has relay-upload cost at most

$$\left(\frac{n}{k_{1,s}} + \frac{n}{k_{1,s}k_{2,s}} \right) |M|$$

up to lower-order terms. By Lemma 5.3, $k_{1,s} \in \{k_1 - 1, k_1\}$ and $k_{2,s} \in \{k_2 - 1, k_2\}$. If $k_{1,s} = k_1$ and $k_{2,s} = k_2$, the variant slightly lowers the sender-upload cost while leaving the non-sender relay-upload bound unchanged. If $k_{1,s} = k_1 - 1$, the larger fragments can increase both sender-upload cost and the relay-upload cost of the other validators. If $k_{1,s} = k_1$ but $k_{2,s} = k_2 - 1$, the sender-upload cost still decreases slightly, while the recovery term in the other validators' relay-upload cost increases. When $k_{2,s} = 1$, the recovery code is a repetition code, and recovery degenerates to

direct repair. If some validator has stake greater than $W/3$, then $\nu_{1/3} = 1$, and this happens no matter who the sender is: a sender P_s with $w_s > W/3$ uses Balanced MiniCast with $k_2 = 1$, and every other sender has $k_{2,s} = 1$.

6 Empirical comparison

Table 1 compares the constructions on validator-stake snapshots of eight networks, taken in May and June 2026 from validator dashboards and network APIs. Appendix C lists the values.

6.1 Virtualization baselines

We compare against Proportional CT and Proportional MiniCast.

Proportional CT. Proportional virtualization assigns each physical validator an integer number of virtual validators and one erasure-code fragment to each virtual validator. Let $v_i \in \mathbb{Z}_{\geq 0}$ be the number of virtual validators controlled by P_i , and write

$$V := \sum_{i=1}^n v_i.$$

For $S \subseteq [n]$, write

$$v(S) := \sum_{i \in S} v_i.$$

Use a (V, k) erasure code, with one fragment assigned to each virtual validator. The physical validators still use the exact stake-weighted voting guards from Section 3; the virtual validators are used only for fragment allocation. The required coding condition is

$$w(S) > W/3 \implies v(S) \geq k.$$

As computed in Section 1, as $V \rightarrow \infty$ one may take $v_i \approx p_i V$ and $k \approx V/3$, giving maximum relay-upload cost approximately

$$3np_{\max}|M|.$$

Swiper weight qualification. Swiper [15] gives a systematic way to choose a virtualization satisfying a weighted threshold condition without requiring very many virtual validators. Swiper calls the virtual validators **tickets**; we continue to denote the virtualization by v and their total number by V . For weight qualification (WQ) at stake threshold $1/3$, one may choose any $\beta_n < 1/3$ and obtain

$$w(S) > W/3 \implies v(S) > \beta_n V.$$

Here β_n is Swiper’s notation; it does not depend on n . Therefore any

$$k \leq \lfloor \beta_n V \rfloor + 1$$

satisfies the required coding condition $w(S) > W/3 \implies v(S) \geq k$. The Swiper paper instantiates this with a $(V, \lceil \beta_n V \rceil)$ erasure code, assigning one fragment to each ticket. Choosing β_n closer to $1/3$ improves the erasure-code rate, but in general requires more tickets. The example in [15] uses

$\beta_n = 1/4$; for direct comparison we do the same, and use a $(V, \lceil V/4 \rceil)$ erasure code. (When $4 \mid V$, the bound above would allow $k = V/4 + 1$; we keep $\lceil V/4 \rceil$ to match [15].) Write $v_{\max} := \max_i v_i$. Validator P_i holds v_i fragments, so the maximum relay-upload cost is

$$\frac{nv_{\max}}{\lceil V/4 \rceil} |M|.$$

To choose the tickets, we use Swiper’s family of assignments, $v_i = \lfloor s w_i + 2/3 \rfloor$ for $s > 0$, where validators that gain a ticket at the same s do so in input order. Swiper finds a local minimum in this family by binary search; we instead select the assignment with the smallest V that satisfies the WQ condition with $\beta_n = 1/4$.

Proportional MiniCast. Proportional virtualization applies to MiniCast as well. The main code must satisfy $w(S) > 2W/3 \Rightarrow v(S) \geq k$, so as $V \rightarrow \infty$ one may take $k \approx 2V/3$, and the recovery code is a (V, k') erasure code with $k' \approx V/3$. A fragment then has size approximately $3|M|/(2V)$, and a mini-fragment has size approximately $9|M|/(2V^2)$. Validator P_i sends its approximately $p_i V$ fragments to all n validators, and sends approximately $p_i V \cdot p_j V$ mini-fragments to each validator P_j . Since $\sum_j p_j = 1$, its relay-upload cost is approximately $(3n/2 + 9/2) p_i |M|$, and the maximum relay-upload cost is approximately

$$\left(\frac{3n}{2} + \frac{9}{2} \right) p_{\max} |M|.$$

6.2 Results

Table 1 reports the coefficients of $|M|$ in the leading terms. It does not include validation paths, tags, or short control messages; for Balanced MiniCast these appear in Theorem 5.2.

Table 1. Leading coefficients of maximum relay-upload cost.

Here $p_{\max}$ is the largest stake fraction, and ν_{θ} is the size of the smallest validator set of stake greater than θW . Proportional CT: $3np_{\max}$ in the limit $V \rightarrow \infty$, and $nv_{\max}/\lceil V/4 \rceil$ with the Swiper WQ tickets of Section 6.1, where V and $v_{\max}$ are as defined there. Balanced CT: $n/\nu_{1/3}$. Proportional MiniCast: $(3n/2 + 9/2) p_{\max}$ in the limit $V \rightarrow \infty$. Direct repair: $2n/\nu_{2/3} - 1$ in the worst case. Balanced MiniCast: $n/\nu_{2/3} + n/(\nu_{2/3}\nu_{1/3})$.

Network	n	$p_{\max}$	$\nu_{1/3}$	$\nu_{2/3}$	CT			MiniCast		
					Proportional			Direct		
					$V \rightarrow \infty$	WQ ($V, v_{\max}$)	Balanced	Proportional	repair	Balanced
Monad	200	12.0%	23	94	72.21	82.76 (231, 24)	8.70	36.65	3.26	2.22
Kava	56	27.5%	2	6	46.14	56.00 (39, 10)	28.00	24.31	17.67	14.00
Sui	127	3.0%	19	49	11.35	13.80 (182, 5)	6.68	5.81	4.18	2.73
dYdX	31	13.0%	5	15	12.05	11.63 (31, 3)	6.20	6.61	3.13	2.48
Sei	40	7.1%	8	18	8.50	8.57 (55, 3)	5.00	4.57	3.44	2.50
Osmosis	70	9.2%	7	22	19.31	22.27 (87, 7)	10.00	10.07	5.36	3.64
Injective	50	6.8%	7	18	10.20	11.11 (70, 4)	7.14	5.41	4.56	3.17
Neutron	16	8.8%	5	9	4.24	5.33 (11, 1)	3.20	2.51	2.56	2.13

The table forms a two-by-two comparison: Proportional versus Balanced, for CT and for MiniCast. The gaps vary substantially with the stake distribution. On Monad, assigning one fragment

per validator reduces the maximum relay-upload cost from $72.21|M|$ to $8.70|M|$ for CT, a factor of about 8.3, and from $36.65|M|$ to $2.22|M|$ for MiniCast, a factor of about 16.5. The gain is larger for MiniCast because its recovery mechanism lets the main code use $\nu_{2/3}$ rather than $\nu_{1/3}$: on Monad, $\nu_{2/3} = 94$ is more than four times $\nu_{1/3} = 23$, whereas proportional virtualization gains only a factor of two from the higher stake threshold. Overall, Balanced MiniCast improves on Proportional CT by a factor of about 32.5 on Monad. Even on Neutron, where the gaps are smallest, the corresponding reductions are from $4.24|M|$ to $3.20|M|$ and from $2.51|M|$ to $2.13|M|$. Kava is the hardest case, because $\nu_{1/3} = 2$; nevertheless, Balanced MiniCast gives $14|M|$, compared with $24.31|M|$ for Proportional MiniCast and $46.14|M|$ for Proportional CT, or $56|M|$ with the WQ tickets. Direct repair is simpler, and in Locher’s protocol it comes with better round complexity. Its worst-case relay-upload cost exceeds that of Balanced MiniCast on all eight distributions, by a factor between 1.2 and 1.5. In addition, a lagging validator may receive twice as much fragment data as the others (Section 5.5). The two costs for Proportional CT are generally of comparable magnitude, and their ordering can reverse slightly on a particular distribution, as on dYdX.

Balancing relay traffic can increase total relay-upload cost (Section 1). Under proportional virtualization, in the limit $V \rightarrow \infty$, the total is about $3n|M|$ for CT and $(3n/2 + 9/2)|M|$ for MiniCast, whatever the stake distribution; for Balanced CT and Balanced MiniCast it is n times the entry in the table. For MiniCast, the total increases by a factor of about 1.5 on Monad and 8.9 on Kava, and by a factor between 1.2 and 2.3 on the other six. For CT, the corresponding factors are 2.9, 9.3, and between 1.1 and 3.3. Section 7 shows how to lower these factors.

7 Between proportional and balanced

Balanced CT and Balanced MiniCast assign one fragment to each validator. For MiniCast, this applies to both the main and recovery codes. For each code, this minimizes the maximum relay-upload cost, but can increase the total relay-upload cost (Section 6.2). Proportional CT and Proportional MiniCast keep the total low, but not the maximum. In this section we consider intermediate fragment assignments that trade total relay-upload cost against maximum relay-upload cost. The protocols run as Proportional CT and Proportional MiniCast do, but with a virtualization that need not be proportional to stake. We say how such a virtualization is found and verified. We then report what we found on the eight distributions, and explain why the gain differs so much between them.

Allocations. Fix a stake threshold θ , and let $v = (v_1, \dots, v_n)$, V , $v(S)$ and $v_{\max}$ be as in Section 6.1. An **allocation** is a pair (v, k) of a virtualization v and a reconstruction threshold k such that

$$w(S) > \theta W \implies v(S) \geq k.$$

The sender uses a (V, k) erasure code and assigns v_i fragments to P_i . One fragment per validator is the allocation $v_i = 1$, $k = \nu_\theta$; proportional virtualization takes $v_i \approx p_i V$ and $k \approx \theta V$.

Protocols. We call CT with an allocation (v, k) for $\theta = 1/3$ **Allocation-based CT**. It runs as Proportional CT does (Section 3), with v in place of a proportional virtualization. In **Allocation-based MiniCast**, the main code has an allocation (v, k_1) for $\theta = 2/3$, and the recovery code has an allocation (u, k_2) for $\theta = 1/3$: each of the $V_1 := \sum_i v_i$ fragments is encoded with a (V_2, k_2) erasure code, where $V_2 := \sum_i u_i$, and P_i holds u_i mini-fragments of each fragment. It runs as Proportional

MiniCast does (Section 6.1), except that u may differ from v . The protocols studied so far are special cases. In Proportional CT and Proportional MiniCast, the virtualization is proportional to stake. Balanced CT is the case of one fragment per validator, $v_i = 1$ and $k = \nu_{1/3}$. Balanced MiniCast is the case $v_i = u_i = 1$, $k_1 = \nu_{2/3}$, $k_2 = \nu_{1/3}$. When $\nu_{1/3,s} > 0$, sender-aware CT is the case $v_s = 0$ and $v_i = 1$ for $i \neq s$, with $k = \nu_{1/3,s}$. Under the same condition, sender-aware MiniCast is the case $v_s = u_s = 0$ and $v_i = u_i = 1$ for $i \neq s$, with $k_1 = \nu_{2/3,s}$ and $k_2 = \nu_{1/3,s}$. The following rules apply to all these variants, including Proportional CT and Proportional MiniCast.

The v_i fragments of P_i , with their Merkle validation paths, form a **bundle**. A bundle is certified if all its fragments are. The sender sends P_i its bundle, and P_i relays it, in its echo in CT and in its vote in MiniCast; a message that carries a bundle counts toward a guard only if the bundle is certified. Otherwise a Byzantine validator could add its stake to a guard while supplying fewer fragments than the allocation condition assumes. In MiniCast, a confirm message from P_j to P_i likewise carries a bundle: for each of the v_i fragments of P_i , the u_j mini-fragments that P_j holds. Summed over i , these are $u_j V_1$ mini-fragments. An empty bundle is the value NULL of Section 5.6, and is treated as certified. With these rules, a vote set of stake greater than $2W/3$ supplies at least k_1 certified fragments, and a confirm set of stake greater than $W/3$ supplies at least k_2 certified mini-fragments of each fragment to be recovered. The proofs of Theorem 4.1 and Theorem 5.1 use the code parameters only in this way, and apply as before.

Costs. Relaying the v_i fragments of an allocation (v, k) to all n validators costs P_i , in the leading term, $(nv_i/k)|M|$. We call $nv_{\max}/k$ the **maximum cost** of the allocation, and V/k its **average cost**, both in units of $|M|$. In Allocation-based CT, these are the maximum and the average relay-upload cost over the validators. By Theorem 3.1, every allocation has maximum cost at least n/ν_θ , and one fragment per validator attains it.

In Allocation-based MiniCast, validator P_i sends its v_i fragments to all n validators, and u_i mini-fragments of each of the V_1 fragments, so its relay-upload cost is, in the leading term,

$$\left(\frac{nv_i}{k_1} + \frac{u_i V_1}{k_1 k_2} \right) |M|.$$

Write $A_1 := V_1/k_1$ and $A_2 := V_2/k_2$ for the average costs of the two allocations. The average cost of the protocol is

$$A_1 \left(1 + \frac{A_2}{n} \right).$$

For a fixed recovery allocation, the second term in each validator's cost is proportional to A_1 , so lowering the main-code average cost lowers that term for every validator. The maximum cost of the protocol is therefore not the maximum cost of the main-code allocation, and Balanced MiniCast need not minimize it.

Finding allocations. Given the stake distribution and θ , and a bound on the maximum cost, we want an allocation with the lowest average cost. The number V of fragments is a secondary concern: more fragments allow a lower average cost, with diminishing returns, but make the code longer, so a bound on V is a further parameter. Both finding an allocation and using it rest on computing, for a given virtualization v , the largest valid threshold

$$k^*(v) := \min\{v(S) : w(S) > \theta W\} = V - \max\{v(T) : w(T) < (1 - \theta)W\},$$

since a set has stake greater than θW exactly when its complement has stake less than $(1 - \theta)W$. The maximum is a knapsack problem with small integer values, which dynamic programming over the values of $v(T)$ solves exactly, using $O(nV)$ operations on stakes; Swiper [15] validates its tickets in the same way.

We find allocations by adapting Swiper. Write the maximum-cost bound as bn/ν_θ , where $b \geq 1$; we call $b - 1$ the **concession**. Let $V_{\max} \geq n$ be the bound on V ; an allocation within both bounds is **admissible**. For a cap m and a scale $s > 0$, let

$$v_i := \min(m, \lfloor s p_i + 1 - \theta \rfloor);$$

without the cap, this is Swiper’s family of ticket assignments for the allocation condition. We consider every integer cap

$$1 \leq m \leq \left\lfloor \frac{bV_{\max}}{\nu_\theta} \right\rfloor.$$

Indeed, every admissible allocation satisfies

$$v_{\max} \leq \frac{b k^*(v)}{\nu_\theta} \leq \frac{bV_{\max}}{\nu_\theta}.$$

Every vector v in this family can also be generated with cap $m = v_{\max}$. For each cap, we increase s from 0, one fragment at a time, scanning only allocations with $V \leq V_{\max}$, and stop at the first v with $k^*(v) \geq 1$ for which $(v, k^*(v))$ has maximum cost within the bound. Among the allocations so obtained, together with one fragment per validator, we take the one of lowest average cost. Finally, we visit the validators from lightest to heaviest, removing fragments from each one at a time as long as $k^*(v)$ does not fall. We try light validators first because they contribute less stake to qualifying sets. We call this procedure **Capped Swiper**. It resolves all ties deterministically as specified in the accompanying artifact [17]. The direct implementation can be accelerated substantially without changing the allocations it returns:

- Consecutive caps share an initial part of the same Swiper scan, so this prefix need not be processed again. After the heaviest validator receives its m th fragment, $v_{\max} = m$ for the rest of the cap- m scan. Admissibility then reduces to requiring $k^*(v)$ to reach a fixed threshold. Since $k^*(v)$ can only increase as fragments are added, this predicate is monotone on the remaining suffix, so binary search finds the same first admissible point as the direct linear scan. This monotonicity is a property of the capped suffix; the full Swiper validity predicate need not be monotone.
- At each binary-search probe, we need only decide whether $k^*(v)$ reaches the threshold required by the current cap. A bounded knapsack DP performs this decision without computing the full value of $k^*(v)$. When a failed probe returns a qualifying set with too few fragments, we cache that set: it certifies later scan points as failures until it has accumulated enough fragments, often avoiding further DP calls.
- During the final removal of fragments, validators are processed from lightest to heaviest while the target value of $k^*(v)$ remains fixed. Rather than recomputing a knapsack DP after each proposed fragment removal, we precompute suffix knapsack tables for validators not yet processed and incrementally maintain a prefix table for validators already shaved. Combining these tables lets us test proposed fragment counts for the current validator using previously computed DP information.

The accompanying artifact gives correctness arguments and implementation details. For fixed b and $V_{\max} = O(n)$, the worst-case cost is $O(n^3 \log n / \nu_\theta)$ arithmetic operations. With our Python/NumPy implementation, on one core of an AMD Ryzen 9 7950X, the two Monad allocations used by MiniCast with a 10% concession and at most $4n$ fragments per code take about 15 ms. Since one fragment per validator is among the candidates, the allocation selected never has a higher average cost than that, and its maximum cost is within the bound, whatever the stake distribution. Beyond that we have no guarantee: Swiper’s bound on the number of tickets relies on proportionality, which the cap gives up.

All validators must use the same allocation. Since Capped Swiper is deterministic, every validator can compute it from the stake distribution. Alternatively, since every validator can verify an allocation by computing $k^*(v)$, it can be computed by any means and then checked by all: a poor allocation costs bandwidth, not security. The allocation depends on the stake distribution and on θ , but not on the sender. The allocations reported below, and programs that compute and verify them, are available [17].

Recomputing allocations. The stake distribution changes over time (for example, on Monad every few hours), and an allocation for one stake distribution need not be one for the next. Capped Swiper is needed once per stake distribution, not once per broadcast, and every validator can run it for itself. An earlier virtualization v also remains valid for the new stakes, with threshold $k^*(v)$, as long as $k^*(v) \geq 1$; the maximum cost and the average cost both grow as $k^*(v)$ falls. Where the stake distribution changes with every block, v can be kept, with $k^*(v)$ recomputed, until its cost has grown too much. Since all validators must use the same allocation, the rule that decides when to replace v must itself depend only on the stake distributions.

For the sender-aware variants of Section 5.6, Capped Swiper can instead be adapted to compute allocations separately for each possible sender, assigning that sender no fragments and using the sender-specific reconstruction conditions. For fixed stakes and parameters, these allocations can be precomputed and reused across broadcasts. The measured running times suggest that such precomputation may be practical, although we have not benchmarked sender-aware allocation.

Results. Table 2 reports, for each stake distribution, the allocation that Capped Swiper finds with maximum cost at most 10% above the smallest maximum cost n/ν_θ and with $V \leq 4n$, that is, with a code at most four times as long as with one fragment per validator. It also reports the average cost of the allocation that it finds with no such concession, that is, with maximum cost n/ν_θ . For MiniCast we allocate both codes in this way, with $\theta = 2/3$ for the main code and $\theta = 1/3$ for the recovery code, using the same concession and a bound of $4n$ fragments for each; the table reports the average cost $A_1(1 + A_2/n)$ of the whole protocol. We did not try to minimize the maximum cost of MiniCast directly. Nevertheless, a concession of $x \geq 0$ on each allocation raises the protocol’s maximum cost by at most the same fraction over Balanced MiniCast, since $A_1 \leq n/\nu_{2/3}$:

$$\max_i \left(\frac{nv_i}{k_1} + A_1 \frac{u_i}{k_2} \right) \leq (1+x) \left(\frac{n}{\nu_{2/3}} + \frac{n}{\nu_{2/3}\nu_{1/3}} \right).$$

For a concession of 10%, the table also gives the lowest average cost found by an integer-programming search under the same bounds, and a lower bound from real-valued shares, both described below. For MiniCast, both are obtained by combining the values for the two codes using the same formula $A_1(1 + A_2/n)$. Since this formula is increasing in each argument, minimizing

each allocation’s average cost separately minimizes the protocol’s average cost under the separate bounds on the two codes.

Table 2. Average relay-upload cost with a 0% or 10% concession on the maximum cost of each allocation, and at most $4n$ fragments per code.

Leading coefficients of relay-upload cost, in units of $|M|$; all cost entries are average costs. Balanced: one fragment per validator in each code; maximum cost and average cost coincide. 0% and 10%: the maximum cost of each allocation is n/ν_θ , or at most 10% above that. Lemma bnd: a lower bound for the 0% setting, derived below. ILP: the lowest average cost found by the integer-programming search with at most $4n$ fragments per code; in the entries marked * the search did not finish, and the value is the lowest that we know of. LP bnd: the lower bound from real-valued shares, with no fragment-count bound. Proportional: the average cost under proportional virtualization, in the limit $V \rightarrow \infty$. For MiniCast, the concession applies to both codes, and the entries are average costs of the whole protocol.

Network	Balanced	0%		10%			Proportional
		Capped Swiper	Lemma bnd	Capped Swiper	ILP	LP bnd	
<i>CT</i>							
Monad	8.70	8.26	7.83	5.34	5.34	4.94	3.00
Kava	28.00	5.66	2.00	5.22	5.10	4.79	3.00
Sui	6.68	5.83	4.16	3.53	3.52*	3.20	3.00
dYdX	6.20	4.80	3.20	4.30	4.26	4.09	3.00
Sei	5.00	5.00	5.00	3.91	3.91	3.82	3.00
Osmosis	10.00	9.29	8.71	6.88	6.19	5.88	3.00
Injective	7.14	5.79	4.29	4.00	3.86	3.73	3.00
Neutron	3.20	2.60	2.60	2.60	2.60	2.60	3.00
<i>MiniCast</i>							
Monad	2.22	2.12	2.00	1.86	1.86	1.79	1.52
Kava	14.00	5.92	3.97	4.12	3.86	3.72	1.58
Sui	2.73	2.58	2.34	1.75	1.75*	1.65	1.54
dYdX	2.48	2.31	2.21	1.96	1.96	1.87	1.65
Sei	2.50	2.16	1.81	1.82	1.82	1.72	1.61
Osmosis	3.64	2.49	1.38	2.01	1.97*	1.85	1.56
Injective	3.17	2.29	1.09	1.86	1.85	1.77	1.59
Neutron	2.13	1.68	1.68	1.68	1.68	1.68	1.78

How much can be gained without raising the maximum cost at all depends on the stake distribution. An allocation with the smallest maximum cost has a lower average cost than one fragment per validator only if some validators hold fewer fragments than others. A set of ν_θ validators of stake greater than θW must hold enough fragments to reconstruct, which it can do only if each of its members holds as many as the maximum cost permits. So only validators that belong to no such set can hold fewer. Let F_θ be the set of these validators. Equivalently, $P_i \in F_\theta$ if and only if w_i plus the stake of the $\nu_\theta - 1$ heaviest other validators is at most θW .

Lemma 7.1. *If an allocation (v, k) has maximum cost n/ν_θ , then $v_i = k/\nu_\theta$ for every $P_i \notin F_\theta$. In particular, $V \geq (n - |F_\theta|) k/\nu_\theta$.*

Proof. Since $n v_{\max}/k = n/\nu_\theta$, every validator holds at most k/ν_θ fragments. Let S be a set of ν_θ validators of stake greater than θW . Its members hold at least k fragments in total, so each holds exactly k/ν_θ . $\square$

Thus an allocation with the smallest maximum cost has average cost at least $(n - |F_\theta|)/\nu_\theta$: compared with one fragment per validator, it lowers the total by at most the fraction $|F_\theta|/n$. If the maximum cost may exceed n/ν_θ by a factor $1 + x$, the same argument gives only $v_i \geq (1 - (\nu_\theta - 1)x)k/\nu_\theta$ for $P_i \notin F_\theta$, which, when $\nu_\theta > 1$, is vacuous once $x \geq 1/(\nu_\theta - 1)$. A small concession on the maximum cost can therefore remove this limit.

Write $L_\theta := (n - |F_\theta|)/\nu_\theta$ for the lemma’s lower bound. The 0% lemma bound in Table 2 is $L_{1/3}$ for CT and $L_{2/3}(1 + L_{1/3}/n)$ for MiniCast. These bound the 0% Capped Swiper column from below; the 10% LP bound instead bounds both 10% columns. The two bounds come from different arguments, so the lemma bound can be weaker even though it permits no concession. Table 1 also reports Proportional CT with the finite Swiper allocation of Section 6.1; on Monad it has maximum cost 82.76 and average cost 3.98.

To assess Capped Swiper, we also searched for allocations by integer linear programming. For a fixed threshold k and a cap m on the v_i , we minimize V subject to $v_i \leq m$ and the allocation condition. With k fixed, this minimizes the average cost V/k among allocations with maximum cost at most nm/k . This is an integer linear program with one constraint for every set of stake greater than θW . There are too many constraints to list, so we add them as needed, using a standard technique (constraint generation): we start with a few of them, and whenever the solver proposes a virtualization, the dynamic program above either finds a constraint that it violates, which we then add, or certifies that there is none. The result is optimal for the full program, since every constraint added is one of its own. The stakes enter only through the dynamic program, which uses exact integer arithmetic. We searched over pairs (k, m) , and solved the instances reported in Table 2 with the open-source solver SCIP [8]. The search took hours, and for $V \leq 4n$ some solves did not finish. We also computed the least average cost over real-valued shares $y_i \geq 0$ in place of v_i/k , subject to $y_i \leq c$ for a cap c , by linear programming, adding constraints in the same way. This value is a lower bound on the average cost of every allocation with maximum cost at most nc , whatever its number of fragments. Without the cap, the linear program is that of Bammert et al. [2] for the sets of stake greater than θW . The value of this uncapped program is within 0.02% of $1/\theta$ on all the distributions except Neutron, so on those distributions proportional virtualization essentially minimizes the average cost.

The 0% columns show how much can be gained without raising the maximum cost of either allocation. For CT on Sei, the lemma bound is 5.00, and Capped Swiper attains it. On Monad, the lemma bound is 7.83 against the balanced cost 8.70, so the average cost can fall by at most 10%; Capped Swiper gives 8.26, a fall of 5%. The lemma need not be tight: for CT on Kava it gives 2.00, below even the 10% LP bound of 4.79. A concession of 10% allows the average cost on Monad to fall from 8.26 to 5.34, while on Kava it falls only from 5.66 to 5.22. For Neutron, the lemma bound is attained for both protocols, and Capped Swiper gains nothing from the concession.

A maximum cost within 10% of the smallest possible for each allocation is compatible with a much lower total cost. On Monad, Balanced CT has maximum and average cost 8.70, and Proportional CT has maximum cost 72.21 and average cost 3.00; CT with Capped Swiper allocations has maximum cost 9.52 and average cost 5.34, against an LP bound of 4.94. For MiniCast, the balanced protocol has 2.22, the proportional one 36.65 and 1.52, and the protocol with Capped Swiper allocations 2.42 and 1.86, against an LP bound of 1.79. Over the eight distributions, the average cost falls by between 19% (Neutron) and 81% (Kava) for CT, and by between 16% (Monad) and 71% (Kava) for MiniCast, while the maximum cost of MiniCast is at most 10% above that of Balanced MiniCast, as proved above. On Kava and Neutron the maximum cost of MiniCast falls

as well: on Kava, Balanced MiniCast costs every validator $14|M|$, whereas MiniCast with Capped Swiper allocations has maximum cost $12.26|M|$ and average cost $4.12|M|$. The increase in total cost reported in Section 6.2 shrinks accordingly: for MiniCast it falls from a factor of 8.9 to 2.6 on Kava and from 1.5 to 1.2 on Monad; for CT it falls from 9.3 to 1.7 on Kava and from 2.9 to 1.8 on Monad.

The average cost with a 10% concession is within 11% of the LP bound in every row except CT on Osmosis, where it is 17% above it. Much of what remains is due to the fragment-count bounds, not to Capped Swiper: under the same bounds, the integer-programming search found a lower average cost in 11 of the 16 rows, and by more than 5% only for CT on Osmosis (10%) and for MiniCast on Kava (7%). The codes are long: the allocations found with a concession of 10% use between $0.8n$ and almost $4n$ fragments per code; on Monad, CT has 785, of which the heaviest validators hold 7 each. With at most $2n$ fragments per code, the average cost is between 0% and 23% higher. The fragment-count bounds for the two codes of MiniCast can be chosen independently. An implementation may use a smaller bound for the recovery code to reduce authentication and per-fragment processing overhead, at the cost of a potentially higher average relay-upload cost.

In summary, the proportional protocols tend to have the lowest total cost, but a maximum cost that grows with the largest stake. The balanced protocols have a much lower maximum cost, the same for every validator whatever its stake: on the eight distributions studied, it is lower than that of the proportional protocols by a factor of up to 8.3 for CT and 16.5 for MiniCast. They also tend to have the highest total cost. With Capped Swiper allocations, the protocols have a much lower total cost than their balanced counterparts on the distributions studied, and their maximum cost is at most 10% higher. Swiper and Capped Swiper search for a virtualization; the balanced protocols need no such search.

8 Related work

Bracha introduced reliable broadcast using authenticated channels alone [3], with $O(n^2|M|)$ communication for large messages. Cachin and Tessaro introduced erasure-coded asynchronous verifiable information dispersal, reducing the leading term to $O(n|M|)$ up to lower-order authentication data [4].

Locher improved the leading constant by using a higher reconstruction threshold and direct repair of missing fragments [9]. MiniCast uses a two-level erasure code to lower the worst-case leading constant further [10]. The present paper adapts this line of work to stake-weighted quorums using $\nu_{2/3}$ and $\nu_{1/3}$ as reconstruction thresholds rather than a stake-proportional virtualization. Erasure-coded dissemination of the same kind is used in the atomic broadcast protocol of [13] and in the reliable broadcast protocols of [11], and we expect these protocols to benefit in the same way in the stake-weighted setting.

Bammert et al. [2] introduce monotone erasure codes, in which fragments may have different sizes and every set in a given family, the access structure, can reconstruct. They use these codes to generalize CT from threshold quorums to arbitrary Byzantine quorum systems, and note that the method extends to related protocols, including MiniCast. Stake-weighted quorums are a special case, which they do not treat separately. Their objective is total fragment size, and their protocol costs are measured by total communication. Their Section 6 construction minimizes total fragment size by linear programming, without a cap on a node’s share. Their Section 6.2 gives the uniform construction used by Balanced CT and the corresponding per-node fragment-size lower bound in

the course of bounding total storage overhead. These elementary bounds characterize our primary objective, maximum relay-upload cost: the uniform construction is optimal in the leading term, although it need not minimize total fragment size. Their observation about MiniCast does not analyze how stake distribution affects its bandwidth costs. We show that changing the main-code stake threshold from $1/3$ to $2/3$ can increase the reconstruction threshold from $\nu_{1/3}$ to $\nu_{2/3}$ by substantially more than a factor of two, and quantify the resulting per-validator relay-upload costs on real stake distributions. In [Section 7](#), we seek low-total-cost allocations subject to bounds on maximum cost and on the number of fragments. The uncapped fractional problem is their linear program; Capped Swiper addresses the constrained allocation problem.

In the unweighted setting, a reliable broadcast protocol is called *balanced* if every party, including the sender, incurs about the same communication cost [\[1\]](#). MiniCast achieves this with the technique that our sender-aware MiniCast variant adapts. Our notion of bandwidth balance is different: it asks that a validator’s relay-upload cost not grow with its stake.

Replacing cardinality guards by stake guards adapts the voting logic to weighted fault assumptions, but does not by itself determine how much fragment data the sender should disperse to each validator. Swiper gives a concrete way to choose a virtualization, including the WQ construction used in [Section 6.1](#) [\[15\]](#); Zinovyev [\[16\]](#) gives variants that use fewer tickets. Capped Swiper ([Section 7](#)) adapts Swiper to a related problem, with a cap on each validator’s tickets and a free reconstruction threshold. Walrus [\[5\]](#), a deployed storage network, uses proportional virtualization directly: it has a fixed number of shards, currently 1000, which are assigned to storage nodes in proportion to their stake, and each shard stores its own fragments of every blob. In a storage network, load in proportion to stake is by design.

9 Conclusion

We have shown that stake-weighted reliable broadcast need not inherit the communication imbalance of proportional virtualization. In the constructions studied here, stake determines which validator sets must suffice for reconstruction, while a validator’s relay-upload cost can be independent of its stake. Balanced MiniCast realizes this principle with a main code governed by $\nu_{2/3}$ and a recovery code governed by $\nu_{1/3}$. On skewed stake distributions, $\nu_{2/3}$ can far exceed $2\nu_{1/3}$, and on the real-world distributions the reduction in maximum relay-upload cost is substantial. Assigning different numbers of fragments to different validators recovers much of the total cost of balancing, while maximum relay-upload cost stays within 10% of its minimum.

Disclosure of AI use

Large language models (various versions of Claude and ChatGPT) were used to help brainstorm ideas, to assist in writing text and code, and to proofread. The authors take full responsibility for the content of this paper.

References

1. Nicolas Alhaddad, Sourav Das, Sisi Duan, Ling Ren, Mayank Varia, Zhuolun Xiang, and Haibin Zhang. Balanced Byzantine reliable broadcast with near-optimal communication and improved computation. In *PODC 2022*, pages 399–417. ACM, 2022. Also available as *Cryptology ePrint Archive, Paper 2022/776*.

2. Vivien Bammert, Annalisa Cimatti, Orestis Alpos, Giuliano Losa, and Christian Cachin. Monotone Erasure Codes. *arXiv:2605.22426*, 2026.
3. Gabriel Bracha. Asynchronous Byzantine agreement protocols. *Information and Computation*, 75(2):130–143, 1987.
4. Christian Cachin and Stefano Tessaro. Asynchronous verifiable information dispersal. In *24th IEEE Symposium on Reliable Distributed Systems (SRDS 2005)*, pages 191–202, 2005.
5. George Danezis, Giacomo Giuliani, Eleftherios Kokoris-Kogias, Markus Legner, Jean-Pierre Smith, Alberto Sonnino, and Karl Wüst. Walrus: An Efficient Decentralized Storage Network. *arXiv:2505.05370v4*, 2026. To appear in *ACM CCS 2026*.
6. Eigen Labs. EigenDA Spec: Assignment. <https://layr-labs.github.io/eigenda/protocol/architecture/assignment.html>.
7. Babak Gilkalaye. RaptorCast: Designing a Messaging Layer. Category Labs blog, May 2025. <https://www.category.xyz/blogs/raptorcast-designing-a-messaging-layer>.
8. Christopher Hojny, Mathieu Besançon, Ksenia Bestuzheva, et al. The SCIP Optimization Suite 10.0. Technical report, Optimization Online, November 2025. *arXiv:2511.18580*.
9. Thomas Locher. Byzantine reliable broadcast with low communication and time complexity. *arXiv:2404.08070v2*, 2025.
10. Thomas Locher and Victor Shoup. MiniCast: Minimizing the Communication Complexity of Reliable Broadcast. *Cryptology ePrint Archive, Paper 2024/571*, 2024. Subsequently published in *EUROCRYPT 2025*, Part V, LNCS 15605, pages 96–115, Springer, 2025.
11. Thomas Locher and Victor Shoup. Improving the Round Complexity of MiniCast. *Cryptology ePrint Archive, Paper 2025/779*, 2025. Subsequently published as *Towards Reliable Broadcast with Optimal Communication and Round Complexity* in *SPAA 2026*, pages 155–166, ACM, 2026.
12. Andrew Miller, Yu Xia, Kyle Croman, Elaine Shi, and Dawn Song. The Honey Badger of BFT Protocols. In *ACM CCS 2016*, pages 31–42, 2016. Also available as *Cryptology ePrint Archive, Paper 2016/199*.
13. Victor Shoup. Sing a song of Simplex. *Cryptology ePrint Archive, Paper 2023/1916*, 2023.
14. Victor Shoup and Nigel P. Smart. Lightweight Asynchronous Verifiable Secret Sharing with Optimal Resilience. *Journal of Cryptology*, 37:27, 2024.
15. Andrei Tonkikh and Luciano Freitas. Swiper: a new paradigm for efficient weighted distributed protocols. *Cryptology ePrint Archive, Paper 2023/1164*, 2023. Also published in *PODC 2024*, DOI 10.1145/3662158.3662799.
16. Anatoliy Zinovyev. Weight reduction in distributed protocols: new algorithms and analysis. *Cryptology ePrint Archive, Paper 2025/1076*, 2025. Also published in *DISC 2025*.
17. Reproducibility artifact for the Capped Swiper experiments. 2026. <https://github.com/victorshoup/bandwidth-balanced-rbc-artifacts/blob/v1/bandwidth-balanced-rbc-artifact.zip>.

A Full Balanced MiniCast protocol

Fig. 1 gives the full protocol summarized in Section 5.2.

$\Pi_{\text{BMiniCast}}$

```

// Sender  $P_s$  with input  $M$ 
 $(\tau, \{(f_i, \pi_i)\}_{i \in [n]}) \leftarrow \text{Encode}(M)$ 
for  $i \in [n]$ : send (disperse,  $\tau, f_i, \pi_i$ ) to  $P_i$ 

// Receiving party  $P_i$ 
 $\text{echoed} \leftarrow \text{voted} \leftarrow \text{confirmed} \leftarrow \text{delivered} \leftarrow \text{false}, \quad \text{acquired} \leftarrow M^* \leftarrow \perp$ 

repeat forever:
  wait until either:
    not  $\text{echoed}$  and for some  $\tau, f_i, \pi_i$ : received (disperse,  $\tau, f_i, \pi_i$ ) from  $P_s \Rightarrow$ 
       $\text{echoed} \leftarrow \text{true}$ 
      if  $(f_i, \pi_i)$  is a certified fragment for  $\tau$  at position  $i$  then
         $\text{acquired} \leftarrow (\tau, f_i, \pi_i)$ 
        send (echo,  $\tau$ ) to  $P_1, \dots, P_n$ 

    not  $\text{voted}$  and for some  $\tau, f_i, \pi_i$ :
       $\text{acquired} = (\tau, f_i, \pi_i)$  and
      received (echo,  $\tau$ ) from parties  $P_j, j \in \mathcal{J}$ , with  $w(\mathcal{J}) > 2W/3 \Rightarrow$ 
         $\text{voted} \leftarrow \text{true}$ 
        send (vote,  $\tau, f_i, \pi_i$ ) to  $P_1, \dots, P_n$ 

    not  $\text{voted}$  and for some  $\tau, \pi_i, \mathcal{J} \subseteq [n]$  with  $w(\mathcal{J}) > W/3$ ,
    and  $\{(\phi_{ij}, \pi_{ij})\}_{j \in \mathcal{J}}$ :
      for all  $j \in \mathcal{J}$ : received (confirm,  $\tau, \phi_{ij}, \pi_{ij}, \pi_i$ ) from  $P_j$ 
      and  $(\phi_{ij}, \pi_{ij}, \pi_i)$  is a certified mini-fragment
      for  $\tau$  at position  $(i, j) \Rightarrow$ 
         $\text{voted} \leftarrow \text{true}$ 
         $f_i \leftarrow \text{Recover}(i, \tau, \pi_i, \{(\phi_{ij}, \pi_{ij})\}_{j \in \mathcal{J}})$ 
        send (vote,  $\tau, f_i, \pi_i$ ) to  $P_1, \dots, P_n$ 

    not  $\text{confirmed}$  and for some  $\tau, \mathcal{J} \subseteq [n]$  with  $w(\mathcal{J}) > 2W/3$ ,
    and  $\{(f_j, \pi_j)\}_{j \in \mathcal{J}}$ :
      for all  $j \in \mathcal{J}$ : received (vote,  $\tau, f_j, \pi_j$ ) from  $P_j$ 
      and  $(f_j, \pi_j)$  is a certified fragment for  $\tau$ 
      at position  $j \Rightarrow$ 
         $\text{confirmed} \leftarrow \text{true}$ 
        if  $\text{Decode}(i, \tau, \{(f_j, \pi_j)\}_{j \in \mathcal{J}})$  returns
         $(M', \{(\phi'_{ji}, \pi'_{ji}, \pi'_j)\}_{j \in [n]})$  then
           $M^* \leftarrow M'$ 
          for  $j \in [n]$ : send (confirm,  $\tau, \phi'_{ji}, \pi'_{ji}, \pi'_j$ ) to  $P_j$ 

    not  $\text{delivered}$  and  $M^* \neq \perp$  and
    received (confirm, ...) from parties  $P_j, j \in \mathcal{J}$ , with  $w(\mathcal{J}) > 2W/3 \Rightarrow$ 
       $\text{delivered} \leftarrow \text{true}$ 
      output  $M^*$ 

```

Fig. 1. The Balanced MiniCast reliable broadcast protocol

B Security proof for Balanced MiniCast

We prove [Theorem 5.1](#).

Proof. Following MiniCast, we establish five properties, P1–P5. Properties P1–P3 establish uniqueness of the tag used by honest votes and confirmations, and P4 and P5 together give correctness and completeness. We say that an honest party **echoes** τ if it sends (echo, τ) ; it **votes for** τ if it sends $(\text{vote}, \tau, \dots)$; and it **confirms** τ if it sends $(\text{confirm}, \tau, \dots)$. A vote is either by echo or by confirmation. We extend this terminology to Byzantine parties, saying that a Byzantine party echoes, votes for, or confirms τ if some honest party receives a corresponding message from that Byzantine party.

P1: If an honest party confirms τ , then parties of total stake greater than $2W/3$ previously echoed τ .

Proof. Consider the point in time where the first honest party confirms τ . It received vote messages for τ from a set $\mathcal{Q}$ with $w(\mathcal{Q}) > 2W/3$. Since $w(B) < W/3$, some honest party must have voted for τ . This honest vote must be a vote by echo: if it were a vote by confirmation, then the voter must have received confirm messages for τ from a set $\mathcal{R}$ with $w(\mathcal{R}) > W/3$, and one of these confirmations would have come from an honest party, contradicting the choice of the first honest confirmation of τ . By the vote-by-echo rule, that honest voter previously received echo messages for τ from parties of total stake greater than $2W/3$.

P2: If an honest party votes for τ , then parties of total stake greater than $2W/3$ previously echoed τ .

Proof. If this is a vote by echo, then the claim holds by definition. If this is a vote by confirmation, then the confirming set has stake greater than $W/3$, and therefore at least one of these confirmations must have come from an honest party. The property follows from P1.

P3: All honest parties that vote or confirm do so for one and the same tag.

Proof. The property follows from P1 and P2 and the fact that each honest party echoes at most one tag. Indeed, if honest parties vote or confirm two tags $\tau \neq \tau'$, then there are echo sets $\mathcal{Q}, \mathcal{Q}'$ with $w(\mathcal{Q}) > 2W/3$ and $w(\mathcal{Q}') > 2W/3$. Their intersection has stake greater than $W/3$, and hence contains an honest party, which would have echoed both tags.

P4: If any honest party outputs a message, then all honest parties eventually output the same message, except with negligible probability.

Proof. Consider the point in time where the first honest party outputs a message. It does so because $M^* \neq \perp$ and it received confirm messages from parties P_j , $j \in \mathcal{Q}$, with $w(\mathcal{Q}) > 2W/3$. This party must previously have confirmed some tag τ , namely the tag for which it set M^* . The honest parties in $\mathcal{Q}$ have total stake greater than $W/3$. By P3, every honest party that confirms or votes must do so for that same tag τ .

All honest parties eventually receive the confirm messages for τ sent by those honest confirmers, whose total stake is greater than $W/3$. The consistency property of *Decode* implies that all honest confirmers under τ decoded the same message M' . For every i , the mini-fragments ϕ_{ij} sent by those honest confirmers belong to the recovery codeword of the i th fragment of M' . By collision resistance, every certified mini-fragment under τ agrees with that codeword. Thus an honest party that has not yet voted eventually receives enough certified mini-fragments of its own

fragment to recover that fragment and vote for τ by confirmation. An honest party that has already voted must have voted for τ , by P3. Thus every honest party eventually votes for τ .

The first successful decode under τ also defines a complete re-encoding of M' whose commitment matches the tag. By collision resistance, every certified fragment under τ agrees with that re-encoding at its position. Whenever the confirm guard fires at an honest party, its vote set has stake greater than $2W/3$ and therefore contains an honest voter. By P3, the common tag of that vote set must be τ , and its certified fragments agree with the successful re-encoding above. Thus *Decode* succeeds, so no honest party can previously have set *confirmed* = **true** while leaving $M^* = \perp$. The honest parties have total stake greater than $2W/3$. Once they all vote for τ , every honest party either has already successfully decoded and confirmed τ , or eventually receives at least k_1 such matching fragments and does so. Every honest party then eventually receives confirm messages from stake greater than $2W/3$ and outputs M' .

P5: If the sender is honest and initiates the protocol, then all honest parties eventually output a message, and any honest party that outputs a message outputs the sender's input message M , except with negligible probability.

Proof. Suppose the sender initiates the protocol, and let τ be the tag generated by the sender. Every honest party eventually receives its certified fragment from the sender and echoes τ . By P1 and P2, any honest vote or confirmation for a tag τ' requires echo support for τ' from stake greater than $2W/3$. Such an echo set contains an honest party, and with an honest sender every honest party echoes only τ . Hence $\tau' = \tau$: every honest vote and confirmation is for the sender's tag. Since the honest parties have total stake greater than $2W/3$, every honest party that has not already voted eventually receives enough echoes and votes for τ by echo. When an honest party receives votes for τ from stake greater than $2W/3$, the set contains at least k_1 certified fragments. These fragments are those generated from the honest sender's input, so *Decode* returns M and the party confirms τ . Eventually all honest parties confirm τ and output M .

That completes the proof. □

C Stake values used in the empirical comparison

The arrays below give the stake values used for Table 1. Values are listed in nonincreasing order, without rescaling. Within each array, read left to right and then top to bottom. This displayed order is also the deterministic input order used for the WQ computation.

Monad ($n = 200$, $W = 15247668648$).

1835065762	327337870	268319403	241177873	207610451
205880649	203387155	155576148	153080995	136803839
124952201	124530482	119323544	116953221	114041403
113250559	107493666	99759960	98829464	97099131
95612176	91725367	91469586	90276414	90100051
87971476	85227000	84897643	84897566	84747912
84224486	82203178	81626111	81220809	81154925
79820056	78670497	78295358	78258772	78221922
78116871	77404618	76420381	76368426	76238326
75699546	75586054	75255024	75221802	75212414
75205042	75194864	75188413	75188130	75153932
75132316	75125252	75122393	75120950	75106879
75100928	75100232	75100189	75100103	75100069
75100060	75100051	75100051	72285281	71218576
68867455	66904959	66711085	66473704	59464097
58726978	58628394	58428425	57328542	57221280
56141891	56061440	56035617	55997063	55940147
55837596	54921391	54889878	54757608	54555289
54060370	53879225	53574132	53562486	53381986
53378246	53365003	53358513	53308980	53302587
53295020	53289888	53289381	53260554	53184946
53171327	53142910	53091129	53065053	53004621
52993082	52965450	52948282	52916642	52897982
52838334	52832249	52829067	52800409	52797068
52793003	52771492	52764744	52756550	52753258
52741348	52730256	52729830	52727345	52720743
52714001	52712151	52699651	52691571	52680048
52680008	52663265	52661039	52660361	52653021
52650449	52649351	52647034	52645939	52645829
52645410	52644727	52640093	52635570	52634929
52634071	52633862	52632885	52631900	52631731
52630405	52627511	52626400	52618514	52618325
52609770	52609225	52602907	52602224	52602101
52600129	52600042	52600006	52600004	52600003
52600003	52600003	52600003	52600001	52600001
52600001	52600001	52600001	52600001	52600001
31319847	30000014	29947006	26348798	25381898
25307047	25305709	25251621	25200003	25170679
25156994	25146581	25138593	25136026	25133434
25121364	25105518	25101675	25100742	25100037

Kava ($n = 56$, $W = 98364889889267$).

27015293017744	20545403876984	7418880383182	6247041850915
3794485389893	3768109431474	3582172557835	3330302479953
3147616543059	2120229987196	1699871326513	1546646524309
1370099407946	1254762723403	1199223815220	1097995437202
1092796685375	844772112357	774063812187	753963867645
663648483900	643792259403	587859537841	390103022128
338228117534	308075878260	301317644482	293507366822
289159588099	264619472942	215785067943	214992042453
213394778023	210515905159	139348362171	100259199413
90517152288	90441508739	84322173163	77731656628
60277742584	32945670313	27109091603	25071748920
22678200896	22329276370	11389143602	11018019149
7111918623	6637665974	5865723756	4886312519
2501162285	1898844615	1224000000	594920275

Sui ($n = 127$, $W = 10000$; voting power).

298	253	215	209	197	185	178	164
164	162	161	161	160	158	157	156
156	154	151	147	146	138	137	135
120	117	114	114	113	112	112	111
111	110	107	105	104	104	102	102
99	97	93	92	91	91	89	88
87	87	87	77	75	74	73	72
71	68	62	61	60	60	59	57
55	54	53	53	52	52	52	52
50	49	49	47	47	47	46	45
45	44	43	43	43	42	42	42
42	41	41	40	40	40	38	38
38	37	37	37	36	36	35	34
34	33	31	30	30	30	28	26
22	22	22	21	21	20	18	18
17	17	13	7	6	4	3	

$dYdX$ ($n = 31$, $W = 228439229012595584294733377$).

29602623098704832749082278	18189984456416367252236564	11706220092656901856894801
9883479842820436411206495	8902060918218572794316308	8638606937017788439258947
8594989840519384615864576	8309166596078208330324098	8201919601960658561807567
8123514706531676098893559	7997338676041206381388698	7877098404738403726035723
7606925219773258887895452	7412452086239216008773619	7305000702974806808987772
7257009715902979734322952	6188064951339836455900041	5636960624847578436498576
5514226391921825721731998	5222063102752496637356765	5076370804591049281525581
4631289642181534708431745	4523231307830838702707387	4502544384176133485786338
4357303854782744624913729	4236898677168943044393858	4232293019214613513891558
3938692956937602414212115	1904660349828961439342967	1681081273976671361355821
1185066774450055809395489		

Sei ($n = 40$, $W = 4267587175433411$).

302426152083242	213056715475724	208602756467190	196507312733282
169641557757118	164224728825399	161065993034206	160442486171377
150049273487461	145583218086549	140410014377526	137327689833211
135355404607920	132213358314993	128895083759730	123994358485173
118562940064644	112603586004066	107889940675577	100572389948701
97837905777701	86831497527473	83389107451653	83091689704419
80503082810647	76877549251832	67500010003786	63977920000000
58831256724841	51440483380192	51187275347255	48673587346329
46243388001632	44208297020320	41274879326480	38859814155207
38119196198143	37095685697266	31608314179242	30611275335904

$Osmosis$ ($n = 70$, $W = 204495907683566$).

18801739300496	12899130408898	12463577242848	8948042137477
8202900864761	6516431716883	5353387345422	5236869249225
4940814180038	4768266680603	4646276736262	4508627814948
4188325513147	4082143949439	4050343423238	4049444297182
3992127221884	3954987057498	3924971857185	3894179475270
3798896002794	3772849828327	3524669987402	3403424779140
3335493680052	3194596442543	3133988868429	3084915431178
2780483563862	2749214983329	2701429135573	2485846959193
2473230405450	2221788186476	2205155144896	2125882028628
2108083900684	1951662683901	1931774092551	1800039179745
1726670429656	1631250543116	1440836011258	1432763098684
1143783445388	1125257538688	1041827990346	1023938752410
975448259776	837274607669	761569717519	757495599389
702517395267	660018243085	618492836057	577202595591
487735064268	483593504190	472734444281	459740004938
433271455327	302660423324	287168414818	215278703442
150049438527	137436008457	133488009645	131758753886
84033744885	54600892822		

$Injective$ ($n = 50$, $W = 57355948389900111951403104$).

3899603040364397687172484	3462496018554682926790503	3438098667298338332002075
2640631711912077465715701	2637915570125923490488958	2361972650893431734273016
2259277172090781995835333	2167394340843793666259500	2003365559388215565054472
1776357352848586367452372	1762263583868407004337261	1743696426100659654185048
1503281337909114937111361	1445381404102549638150740	1371905829617903214096019
1352487208836910860356267	1242216671250655141854618	1170370812957437709110477
1115844579620974322923205	1028151988600000000000000	1025721831866375522078355
1009672599364866518719506	1000121425347654105552138	950752271042944718584523
9180025002100000000000005	916798295805676360822262	844879530997957704818740
752064386764152839091543	724572404277024819571726	692916729188240262174919
651689608009001679776856	631660431063704585393637	612927803583309403580164
550416477317306155475403	508804620654590654329746	505897157636836581615449
458784033928297389944812	432964382540476988500027	425939219815389944386698
409996701520888549137070	400001001233701676767000	399818298181600881294400
380685393244795874287393	341837685243651258493588	305979434330801115573737
300481011846351532087984	299978389583791368960910	269512029263139891731668
200338680028731598409156	50022128824010257074279	

Neutron ($n = 16$, $W = 188571240869554$).

16639597260775	14692642098914	14432251343499	14330222238882
14309326119367	14270341332397	14227798122726	14218487274920
14198058977793	14193737538825	14187379997499	14160723261725
14154034962405	310456780738	126806503914	119377055175