

Multimmit: Extending Blocks for Faster Finality

ANDREW LEWIS-PYE^{1,2}, PATRICK O’GRADY², and BEN CLABBY²

¹London School of Economics, UK and ²Commonware, USA

Abstract

To meet the throughput demands of modern blockchain systems, protocols for State Machine Replication (SMR) increasingly have many processors disseminate blocks of transactions in parallel, with a consensus mechanism then establishing a total ordering on the blocks of all producers. Such designs face a basic choice as to when a block may enter the ordering process. The *certified* (or pessimistic) approach waits until a block’s availability has been attested by a quorum. This is robust, but adds message delays to the latency of every transaction. The *uncertified* (or optimistic) approach lets proposals reference blocks immediately. This achieves low latency in favourable conditions, but degrades rapidly when referenced data is missing and must be fetched on the critical path. Raptr, the state of the art, takes a middle course: leaders propose sequences of batches without requiring that voters hold the corresponding data, each voter supports the longest initial segment of the sequence whose data it does hold, and the protocol finalises a prefix supported by a quorum, so that no processor ever blocks or fetches on the critical path. The remaining weakness is sensitivity to order. If the data behind a single batch early in the sequence is withheld, most voters can support only a short prefix, and the proposal finalises little or nothing, however much available data it references: individual faulty producers can therefore still deny the system its optimistic path.

We present Multimmit, a protocol for $n \geq 5f + 1$ processors that combines a consensus layer requiring one round of voting per view with multi-chain data dissemination. Votes are cast relative to the leader’s proposal, reporting per chain how far the voter can endorse it, and may themselves endorse fresh blocks beyond the proposal. A transaction block disseminated at time t is then ordered by $t + 3\delta$ in expectation and $t + 2\delta$ at best, where δ bounds message delay, these figures being measured from the dissemination of the transaction block itself rather than from the leader’s proposal. Performance degrades gracefully in the presence of faults: a faulty producer can delay only its own chain’s blocks, costing other chains at most a one-view wait for placement in the total ordering. The protocol also provides a strong censorship-resistance guarantee, by which no leader can both finalise its leader block and exclude a fresh, well-circulated block of an honest chain. Consensus objects total tens of kilobytes per view, independent of transaction volume. The protocol thereby combines the strengths of the two standard approaches: its latency is below that of certified designs by roughly two message delays (one bought by the resilience assumption $n \geq 5f + 1$, stronger than the $n \geq 3f + 1$ under which Autobahn and Raptr operate, and one by the new mechanisms of this paper), while it exhibits none of the fragility of uncertified approaches, since nothing is ever fetched on the critical path, and no minority of faulty producers can deny honest chains their fast path.

1 Introduction

State Machine Replication (SMR) is a fundamental primitive for distributed computing that allows a collection of processors to maintain a consistent, shared log of transactions despite the

failure, or even malicious behaviour, of some participants. Originally developed for fault-tolerant systems, SMR has become the algorithmic backbone of modern blockchains and decentralised applications. The core challenge is to ensure that all correct processors agree on the same sequence of transactions (Consistency), while also guaranteeing that new transactions are eventually included (Liveness), even in the presence of Byzantine faults, where corrupted processors may behave arbitrarily.

Protocols for SMR typically operate in the partially synchronous model [28], in which messages may be delayed arbitrarily before an unknown time GST (the Global Stabilisation Time), but are delivered within a known bound Δ thereafter. This model captures realistic network conditions, including temporary partitions, and underpins many widely deployed protocols.

The throughput challenge. In classical leader-based protocols such as PBFT [21] and its many derivatives, the leader of each view assembles transactions into a block and sends it to all other processors, who vote on whether to accept it. A disadvantage of this approach is that the leader’s outgoing bandwidth is then a bottleneck for the whole system. Erasure coding alleviates the bottleneck, the leader dispersing coded chunks that the processors then relay, and is used to this end by Kudzu [65] and Alpenglou [41]. It comes at the price of a data expansion rate, typically around three, so a leader with x bits of transaction data still sends around $3x$ bits. Another response, now standard, is to have *all* processors disseminate transaction data in parallel, with the consensus mechanism operating on small references to that data rather than on the data itself. DAG-based protocols [26, 72, 9] realise this idea with every processor proposing concurrently, though modern designs still organise the ordering of the resulting graph around designated anchor blocks. They also couple dissemination to consensus structurally, since a block enters the ordering only once later blocks reference it, so that placement depends on the continued growth of the graph. Multi-chain designs such as Autobahn [34] retain the leader-based structure, decouple the two concerns, and are the starting point for this paper. There, each processor builds its own chain of transaction blocks, disseminating each block as it is produced, and a leader-based protocol run on small *leader blocks* establishes a total ordering on the transaction blocks of all chains. Transaction data then travels over the network only once, from its producer to everyone else. Provided production is spread across enough producers, the system can then in principle finalise transactions at roughly the rate at which individual processors can send and receive data (Section 9 returns to the accounting).

Certified and uncertified tips. Designs of this kind face a basic choice as to when a transaction block may be referenced by the ordering process. The *certified* (or pessimistic) approach, taken by Autobahn, allows a leader block to reference only transaction blocks whose availability has been attested by a quorum. This is robust, but it prices the certification round trip into the latency of every transaction. Three message delays separate a transaction block from its earliest possible inclusion in a leader block, since the block must reach the attesters, their attestations must return to be assembled into a certificate, and the certificate must reach the leader. The *uncertified* (or optimistic) approach allows proposals to reference transaction blocks immediately, with voters missing the corresponding data expected to fetch it before voting. This achieves low latency in favourable conditions, but degrades badly when data is missing, since fetching then sits on the critical path to consensus. The effect is not hypothetical, and has been measured in the DAG setting: a total message loss rate of 0.05% across the network (one per cent of the egress traffic at five per cent of processors) has been observed to increase the latency of Mysticeti [9],

which references uncertified data, by an order of magnitude [7, 75].

Raptr [75], the current state of the art, takes a middle course. Leaders propose sequences of batches without requiring that voters hold the corresponding data, each voter supports the longest initial segment of the proposed sequence whose data it does hold, and the protocol finalises a prefix supported by a quorum. No processor ever blocks or fetches on the critical path, and whatever is finalised is certainly available. The remaining weakness, which we refer to as *easy spoiling*, is sensitivity to order. If the data behind a single batch early in the proposed sequence is withheld, then most voters can support only a short prefix, and the proposal finalises little or nothing, however much available data it references. A single faulty producer can arrange exactly this by withholding one batch, and k faulty producers, taking turns, can deny the entire system its optimistic path for k consecutive proposals. Raptr mitigates the attack with a reputation mechanism, but the mitigation is reactive, in that each faulty processor spoils before it is blamed, and reputations must be forgiving enough to readmit processors after periods of asynchrony, so that any reset re-arms the same attack.

1.1 Our contributions

We present Multimmit, a protocol that combines multi-chain data dissemination with the consensus skeleton of Minimmit [24], in which a single round of voting suffices for finality under the assumption that $n \geq 5f + 1$, for n the number of processors and f the number that may be faulty.¹ We work in the standard partially synchronous model: messages may be delayed arbitrarily before an unknown time GST (the global stabilisation time), and are delivered within a known bound Δ thereafter. Latencies are stated throughout in terms of δ , the *actual* least upper bound on message delay after GST, which is unknown to the protocol and may be far smaller than Δ . The design is guided by a simple principle, that *a faulty processor other than the leader should only be able to significantly delay the finalisation of transaction blocks in its own chain*. Two mechanisms are used to achieve this.

- *Proposal-relative voting.* The leader proposes tips for every chain, certified or not, and each voter responds with a single vote, reporting per chain how far up the chain it can endorse the proposal. No processor blocks or fetches, a withheld block costs only its own chain’s place in the ordering, and leader blocks are finalised after a single round of voting.
- *Extension voting.* Votes may themselves endorse fresh blocks beyond the proposed tips. For a transaction block to be finalised in a view, it then suffices that the block reach the *voters* before they vote, rather than the leader before it proposes. This removes the wait for the leader’s next proposal from every transaction’s critical path, and reduces latency by a further δ on average, independently of the δ bought by the single round of voting.

The design has three principal benefits, which we quantify and compare with the state of the art in Section 3.

1. *Low latency.* Throughout the paper we measure latency from the dissemination of the transaction block itself, rather than from the leader’s proposal, this being the leg of the

¹ $5f - 1 \leq n$ is necessary and sufficient for finality in one round of voting (see [44] and <https://decentralizedthoughts.github.io/2021-03-03-2-round-bft-smr-with-n-equals-4-f-equals-1/>). As is standard [65, 41, 67], we assume $5f + 1 \leq n$ for simplicity. The assumption is materially stronger than the $n \geq 3f + 1$ under which Autobahn and Raptr operate, and is the price of finality in a single round of voting; Section 3.6 keeps the accounting explicit.

journey that competing designs sometimes leave unmeasured. The distinction matters for user-facing latency. An operator’s API node, running a producer chain of its own, places a user’s transaction directly into consensus, rather than forwarding it and waiting a further hop for its inclusion in some leader’s proposal. We also distinguish the *good case*, in which the network is synchronous and all processors are correct, from the *common case*, in which the network is synchronous and the leader correct but up to f other processors may be faulty. In the good case, a transaction block disseminated at time t is placed in the total ordering by $t + 3\delta$ in expectation and by $t + 2\delta$ at best. The corresponding figure for Raptr is $t + 5\delta$ in expectation. In the common case, an honest chain’s block disseminated at time t is still *finalised* by $t + 3\delta$ in expectation, and finalisation means more than the eventual inclusion that certification alone already provides: the block’s inclusion and the content of its chain’s contribution to the ordering are fixed, and *any* leader block subsequently finalised, by whichever leader and however that leader behaves, must place the block in the total ordering. Placement in the total ordering therefore accompanies finalisation at $t + 3\delta$ unless the block is queued behind a lagging faulty chain, and completes by $t + 5\delta$ in expectation in any case. Raptr’s figures degrade in the common case: faulty producers can deny it its optimistic view exits, stretching views from 2δ to 3δ , so that the same block is first finalised at $t + 5.5\delta$ in expectation, and by easy spoiling that finalisation may in any case order little or nothing. In short, *Multimmit establishes a total ordering that faulty producers cannot spoil before Raptr establishes the ordering that they can.*

2. *Robustness.* A faulty producer can delay only its own chain’s blocks, costing other chains at most the one-view wait, just described, for placement in the total ordering, and easy spoiling has no analogue. Since voters report what they hold rather than fetching what they lack, message loss inflates no critical path.
3. *Censorship resistance.* The protocol provides a guarantee of block inclusion that does not depend on the leader. If a fresh transaction block produced by a correct processor reaches all correct processors before they vote in a view, then either the view finalises nothing at all, or the block’s membership of the eventual ledger is settled in that view and it enters the total ordering by the next finalised leader block, at the latest. A leader wishing to censor an honest producer’s block must therefore suppress its entire view, forgoing whatever rewards finalisation brings.

Specification and analysis. We give a complete formal specification of the protocol, with proofs of consistency and liveness, and an accounting of data availability for every block entering the ordering. We also determine the protocol’s extraction thresholds exactly, showing that its rules are optimal at $n = 5f + 1$ and identifying the precise relaxation available for larger n . The specification is written to be implemented, and Section 3.7 calculates the concrete sizes of every object the protocol transmits.

Structure of the paper. Section 2 defines the model. Section 3 develops the protocol informally, compares it quantitatively with Raptr, and collects the guarantees case by case, including for *greedy* Byzantine leaders, meaning those that seek finalisation and avoid provable misbehaviour. Section 4 gives the formal specification, Section 5 the analysis, and Section 6 optimisations, including a backup-proposal extension under which the view of a leader that crashes before proposing finalises as in a correct-leader view. Sections 7–9 describe experiments, related

work and final comments.

2 The setup

The processors. We consider a set $\Pi = \{p_1, \dots, p_n\}$ of n processors. For f such that $5f+1 \leq n$, at most f processors may become corrupted by the adversary during the course of the execution, and may then display Byzantine (arbitrary) behaviour. Processors that never become corrupted are referred to as *correct*.

Cryptographic assumptions. Our cryptographic assumptions are standard for papers on this topic. Processors communicate by point-to-point authenticated channels. We use a cryptographic signature scheme, a public key infrastructure (PKI) to validate signatures, and a collision resistant hash function H . Beyond ordinary signatures, we make use of an aggregate signature scheme (e.g. BLS): signatures on (possibly distinct) messages under (possibly distinct) keys may be aggregated into a single group element, verifiable given the constituent (message, key) pairs, at a cost of one pairing computation per distinct message. We also use two threshold schemes, with thresholds $n - 2f$ and $2f + 1$: each processor thus holds an ordinary signing key and a key-share for each threshold scheme. We assume a computationally bounded adversary. Following a common standard in distributed computing and for simplicity of presentation (to avoid the analysis of negligible error probabilities), we assume these cryptographic schemes are perfect, i.e., we restrict attention to executions in which the adversary is unable to break them.

The partial synchrony model. We consider the standard partial synchrony model, whereby the execution is divided into discrete timeslots $t \in \mathbb{N}_{\geq 0}$ and a message sent at time t must arrive at some time $t' > t$ with $t' \leq \max\{\text{GST}, t\} + \Delta$. We also write δ to denote the (unknown) least upper bound on message delay after GST (noting that δ may be significantly less than the known bound Δ). While Δ is known, the value of GST is unknown to the protocol. The adversary chooses GST and also message delivery times, subject to the constraints already defined. Correct processors begin the protocol execution before GST and are not assumed to have synchronised clocks. For simplicity, we do assume that the clocks of correct processors all proceed in real time, meaning that if $t' > t$ then the local clock of correct p at time t' is $t' - t$ in advance of its value at time t . Using standard arguments, our protocol and analysis can be extended in a straightforward way to the case in which there is a known upper bound on the difference between the clock speeds of correct processors.

Transactions. Transactions are messages of a distinguished form, signed by the environment. Each timeslot, each processor may receive some finite set of transactions directly from the environment. We make the standard assumption that transactions are unique (repeat transactions can be produced using an increasing ‘ticker’ or timestamps).

State machine replication. If σ and τ are sequences, we write $\sigma \preceq \tau$ to denote that σ is a prefix of τ , and say σ, τ are *compatible* if $\sigma \preceq \tau$ or $\tau \preceq \sigma$. If two sequences are not compatible, they are *incompatible*. If σ is a sequence of transactions, we write $\text{tr} \in \sigma$ to denote that the transaction tr belongs to the sequence σ . Each processor p_i is required to maintain an append-only log, denoted $\log_i$, which at any timeslot is a sequence of distinct transactions. We also write $\log_i(t)$ to denote the value of $\log_i$ at the end of timeslot t . The log being append-only means that, for $t' > t$, $\log_i(t) \preceq \log_i(t')$. We require the following conditions to hold in every execution:

Consistency. If p_i and p_j are correct then, for any timeslots t and t' , $\log_i(t)$ and $\log_j(t')$ are

compatible.

Liveness. If p_i and p_j are correct and if p_i receives the transaction tr then, for some t , $\text{tr} \in \log_j(t)$.

As is common for protocols that separate data dissemination from ordering, what the protocol explicitly guarantees for the transaction blocks entering the log is *data availability* (sufficiently many correct processors hold the data of each such block for it to be retrievable); how processors retrieve the data of blocks they are missing is left as an implementation detail. Formally, we thus solve Extractable SMR in the sense of [48]; since the distinction is routine, we do not belabour it.

Blocks, parents, and ancestors. The protocol produces *blocks* of two kinds, defined precisely in Section 4: transaction blocks, which carry sequences of transactions, and leader blocks, which carry none. In each case there are distinguished *genesis* blocks, and each block b other than a genesis block specifies a unique *parent* (by hash value), of which it is a *child*. The *ancestors* of b are b and all ancestors of its parent, while a genesis block has only itself as ancestor. We say b *extends* (or is a *descendant* of) b' if b' is an ancestor of b , and that two blocks are *incompatible* if neither is an ancestor of the other. The *height* of a block is its number of proper ancestors (so that genesis blocks have height 0), and we refer to the greatest block for which a party has received all ancestors on a given chain of transaction blocks as its *tip* of that chain. When the protocol finalises transaction blocks in a given order, processors append the corresponding transactions to their logs, removing any duplicates; the precise extraction of this ordering is specified in Section 4.

3 The intuition

We want a protocol reflecting a commonly used blockchain architecture, in which a designated set of processors is responsible for block production. In general, these block producers might be distinct from the ‘validators’ that carry out consensus. For the sake of simplicity, we assume here that the n validators are the block producers, but one might more generally consider K block producers, for K different from n . In the style of Autobahn, we suppose that each block producer builds its own chain, and consensus must then totally order the blocks from all chains.

Multimmit therefore produces blocks of two kinds. Each processor p_i builds its own chain of *transaction blocks*. Transaction blocks carry transactions but are never themselves the subject of consensus votes. A leader-based protocol, essentially Minimmit, is run on *leader blocks*; leader blocks carry no transactions, but determine a total ordering on transaction blocks. In this section we describe the design informally. We first recall Minimmit, then describe the chain layer, and then build up the consensus layer in two stages, first with voting relative to the leader’s proposals alone, and then with *extension votes*, which sharpen both latency and censorship resistance.

The good and common cases. Throughout the section it will be useful to distinguish two regimes. In the *common case*, the network is synchronous and leaders are correct, while up to f other processors may be faulty. In the *good case*, the network is synchronous and *all* processors are correct. The common case is the most appropriate benchmark for a protocol’s performance, since some faulty processors are to be expected in normal operation, and a protocol that performs well only in the good case is fragile.

3.1 Recalling Minimmit

First, we recall the Minimmit protocol [24].

One round of voting. Minimmit is a view-based protocol. Each view v has a designated leader, who proposes a block b . The proposal specifies as parent a block from an earlier view. Other processors then send signed votes for b to all. Upon receiving $n - f$ votes for b (an *L-notarisation*, ‘L’ for ‘large’), a processor finalises b . This is ‘2-round finality’, one round to send the block and one round of voting, for which $5f + 1 \leq n$ suffices (see Section 1).

View progression. A view whose leader is faulty may never produce an L-notarisation, so processors need a sound licence to move to the next view. To this end, a processor that sees no progress in view v will eventually time out and send a *nullify*(v) message, indicating that it wishes to abandon the view. While no protocol can ensure that every view produces an L-notarisation, Minimmit ensures that every view produces at least one of the following, either of which suffices:

- An *M-notarisation* (‘M’ for ‘mini’) for a view v block b : a set of $2f + 1$ votes for b . This proves that no *other* view v block can receive an L-notarisation, since the two vote sets would share $(2f + 1) + (n - f) - n \geq f + 1$ processors, hence a correct one, and correct processors vote at most once per view. It is therefore safe to enter view $v + 1$ and to build on b .
- A *nullification* for view v : a set of $2f + 1$ *nullify*(v) messages. The rules governing when processors may send *nullify*(v) messages are arranged to ensure that a nullification proves that *no* view v block receives an L-notarisation. It is therefore safe to enter view $v + 1$ and to skip view v entirely.

Arranging that every view really does produce one of the two is the one slightly delicate part of the protocol, and we ask the reader to take this guarantee on trust for now (the rules appear in Section 4). Granting this guarantee, the rest of the protocol and its correctness can be seen directly, as follows.

Anchoring proposals. The leader of view v selects the *greatest* $v' < v$ for which it has received an M-notarisation, and proposes a child of the corresponding block. By the rules for view progression, the leader must then have received nullifications for every view in the open interval (v', v) , since it progressed through each of those views while holding an M-notarisation for none of them. To vote for the proposal, other processors require an M-notarisation for the parent and nullifications for every view in (v', v) , which together prove that no block that could have been finalised is being skipped. If the leader is correct then, during synchrony, this requirement is automatically satisfied, since the leader has received these objects, and every processor forwards each M-notarisation and nullification to all others upon first receipt.

Consistency. These rules already give consistency. Suppose a block b for view v receives an L-notarisation and, towards a contradiction, consider the least view $v' > v$ in which some block b' incompatible with b receives an M-notarisation. (If consistency fails, such a view exists, since an L-notarisation contains an M-notarisation.) Let b'' , from view $v'' < v'$, be the parent of b' . Since at least $f + 1$ correct processors voted for b' , the block b'' has an M-notarisation, and nullifications exist for every view in (v'', v') . Now:

- If $v'' = v$ then $b'' = b$, since b is the only view v block with an M-notarisation. Then b' extends b : a contradiction.
- If $v'' \in (v, v')$ then b'' is compatible with b , by the choice of v' . Since views increase along chains, b'' extends b , so again b' extends b : a contradiction.
- So $v'' < v$. Then a nullification exists for view v itself, contradicting b 's L-notarisation.

Liveness. View progression and the forwarding of M-notarisations and nullifications give liveness. During synchrony, a correct leader selects the greatest v' for which it holds an M-notarisation and proposes. By the time the proposal arrives (strictly, within δ of this time), every correct processor holds the parent's M-notarisation and the interim nullifications. All correct processors therefore vote, and there are at least $n - f$ of them, so the proposal receives an L-notarisation and is finalised.

What carries over. Multimitt inherits this skeleton wholesale: views with one round of voting for finality at $n - f$, every view producing either a notarisation of the smaller kind or a nullification, and proposals anchored as above. There are two substantive changes. First, leader blocks contain no transactions; their role is to coordinate the chains of transaction blocks (described next). Second, the objects playing the role of M-notarisations, which we call V-QCs, certify somewhat more (and are correspondingly larger). We return to this once the chain layer is in place.

3.2 The chain layer

Building n chains. Following Autobahn [34], each processor builds its own chain of transaction blocks, disseminating each block as it is produced. Upon receiving a block b on p_i 's chain, each processor sends a *DA-vote* ('data availability vote', a threshold signature share) for b back to p_i , who forms and disseminates the resulting *DA-certificate* from $n - 2f$ shares.² A DA-certificate ensures *data availability* (at least $n - 3f \geq 2f + 1$ correct processors hold the block, so it can always be recovered) and *uniqueness*, since two certified blocks at the same height of the same chain would give two sets of $n - 2f$ DA-voters intersecting in $n - 4f \geq f + 1$ processors, and correct processors DA-vote at most once per (chain, height).

Pipelining. Here we deviate from Autobahn, which requires each transaction block to contain a certificate for its parent. This puts a full round-trip between consecutive blocks, and so caps the rate at which any single chain can grow. In Multimitt, a producer may instead run up to d blocks ahead of its last certified block (the *pipelining depth*, e.g. $d = 3$), with certificates forming in the background. The price is that DA-votes may now be cast for a block before any certificate for its parent is seen, and the DA-voting rules must be formulated with some care to ensure that certified blocks remain unique and compatible per chain.

²Sending DA-votes to the block producer alone, who then disseminates the certificate, costs one extra message delay in certificate formation. The alternative, in which every processor sends its DA-vote to all, would save that delay but at the cost of cubic communication per transaction block height: n processors sending to n recipients, for each of the n chains. Since (as we shall see) leader blocks may reference transaction blocks before their certificates form, certificate formation is off the critical path, and the extra delay is of little consequence.

The DA-voting rules. The rules are simple to state. Upon receiving a block b of height h on p_i 's chain, p_j considers the greatest height $h' < h$ at which it has seen a DA-certificate for the chain, and DA-votes for b provided:

- p_j has not previously DA-voted for any block of height h on the chain;
- $h - h' \leq d$;
- p_j holds the full sequence of blocks linking the certified block at height h' to b , and has DA-voted for each block in that sequence below b .

We leave the reader to convince themselves (or to await Section 5) that these conditions ensure the compatibility of certified blocks.

3.3 Uncertified tips, and easy spoiling

The latency problem. The simplest way to combine the two layers is the method employed by Autobahn, in which each leader block references, for each chain, the greatest DA-certificate the leader has seen, and finalising the leader block then finalises everything below the referenced tips. This, however, puts three message delays (block, DA-votes, certificate) between a transaction block and its earliest possible inclusion in a leader block. We would like the leader to be able to reference a block *immediately*, with certification proceeding in the background.

Referencing uncertified tips. Referencing a transaction block before it is certified is dangerous, because the reference itself does not prove that the data behind it is available. Autobahn does suggest an optimisation of exactly this kind, whereby leaders may propose uncertified tips (via hash values), and a voter missing the corresponding data holds off voting and fetches it from the leader. The cost is twofold. Timeouts must be inflated to cover the time the leader spends serving such requests, so faulty leaders cause longer delays. Beyond this, the leader may be required to transmit transaction data, on the critical path to voting, at volumes of precisely the order that multi-proposer dissemination exists to avoid. As an example of the latter, suppose that each of $f = \Omega(n)$ faulty producers sends its latest block to the leader alone. If the leader references these tips, then every correct voter is missing all $\Omega(n)$ blocks and must fetch them from the leader before voting. The leader is then forced to send $\Omega(n)$ blocks to each of $\Omega(n)$ participants, i.e. $\Omega(n^2)$ block transmissions through a single uplink, before the view can progress.

Easy spoiling. Raptr [75] addresses the same problem without blocking or fetching on the critical path. There, a leader's proposal specifies a sequence of transaction batches by reference, without requiring that the referenced data be certified, or even that voters hold it. Each voter votes for the longest initial segment of the sequence for which it *does* hold all the referenced data, and the protocol finalises an initial segment that sufficiently many voters support (roughly speaking, the prefix voted for within a quorum). No processor waits or fetches on the critical path, and whatever is finalised is certainly available.

The fragility lies in the sensitivity to order. If the data behind even one batch referenced early in the sequence has been withheld, most voters can vote only for a very short prefix, and the block finalises little or nothing, however much available data it references beyond that point. A single faulty producer can arrange exactly this by withholding one batch, and k faulty producers, taking turns, can spoil k consecutive leader blocks. We refer to this failure as *easy spoiling*. We note that spoiling degrades Raptr rather than halting it, since batches stranded

behind withheld data are eventually included by way of data availability certificates, as in the basic approach. However, this fallback is precisely the slow path that prefix voting exists to avoid, so k faults deny the entire system its optimistic path for k blocks, rather than merely delaying the withheld data. Raptr mitigates the attack with a reputation mechanism, under which producers blamed by $f + 1$ processors have their batches included only via the certified path (batches of merely slow producers are instead reordered towards the ends of proposals). The mitigation is reactive, however (each faulty processor spoils before it is blamed), and reputations must be forgiving enough to readmit processors that were merely slow, for example after a period of asynchrony, so any reset re-arms the same attack. Avoiding the failures of both approaches (Autobahn’s blocking fetch from the leader, and Raptr’s easy spoiling) motivates our guiding principle: *a faulty non-leader processor should only be able to significantly delay the finalisation of transaction blocks in its own chain.* (We make precise the extent to which other chains can be affected at all in Section 3.4 below.)

Our approach, in outline. Multimmit responds with two mechanisms, developed in the next two subsections.

- *Proposal-relative voting.* The leader issues a *chain proposal*, proposing tips for every chain, and each voter responds with a *single* vote, which is now a vector with one coordinate per chain, reporting how far up each chain the voter can endorse the chain proposal.³ No processor blocks or fetches, a withheld block costs only its own chain’s slot, and leader blocks are finalised after a single round of voting.
- *Extension votes.* Votes may themselves endorse fresh blocks beyond the proposed tips. For a transaction block to be included, it need then only reach the *voters* before their next vote, rather than reaching the *leader* before its next proposal, removing the proposal leg (one message delay, plus the wait for a block production event) from the critical path.

Extension votes also yield a form of block inclusion guarantee: if a fresh transaction block produced by a correct processor reaches all correct processors before they vote in a view then, so long as the view finalises any block at all, the block’s membership of the eventual ledger is settled in that view (no competing block can ever be certified), and it enters the total ordering at latest via the next finalised leader block. The content of the leader’s proposal cannot prevent this, since a leader wishing to exclude an honest producer’s block must suppress the entire view, forgoing whatever rewards finalisation brings. (The precise statement, including the sense of ‘fresh’, is the block inclusion theorem of Section 5.) The guarantee is aimed at precisely such processors, motivated to finalise blocks but tempted to censor particular producers or transactions. As we quantify in Section 3.6, the resulting latency is much lower than Raptr’s, in the good case and by a wider margin in the common case. Of the roughly 2δ saved in the good case, only one δ is bought by the assumption $5f + 1 \leq n$ (one round of voting rather than two), while the rest comes from the mechanisms above.

Finalisation and ordering. It will help throughout to keep the following picture in mind, distinguishing the transaction blocks a quorum certificate *finalises* from the portion of the total ordering that can be *extracted* from it. Similar to Minimmit (with V-notarisations now replacing

³The reader may worry that votes thereby become large. Such vectors condense well. A vote that simply endorses every proposed tip can be expressed in constant size, and in general a vote need record little more than its deviations from full endorsement (see Section 3.7).

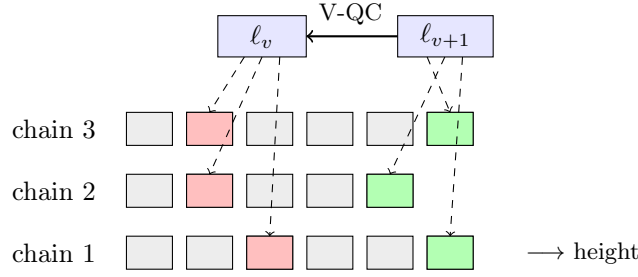


Figure 1: The two layers. Producers extend their own chains of transaction blocks continuously (bottom), while consensus votes only on leader blocks (top), each of which references the previous view’s V-QC (solid arrow) and proposes a tip per chain (dashed arrows), red for ℓ_v and green for ℓ_{v+1} . Each proposal extends the tips of the view it builds on, and chains grow between proposals without waiting for the consensus layer.

M-notarisations), every view produces either a nullification or a *V-notarisation*, a set of $n - f$ view v messages, sufficiently many of which are votes for the view’s leader block. Since votes are now vectors, notarisations are larger objects than their Minimmit counterparts, and the protocol in fact manipulates succinct certificates representing them, *V-QCs* and *L-QCs* respectively. It does no harm to conflate each notarisation with its certificate, and we do so freely below. Section 3.7 gives calculations showing that the certificates remain small.

A leader block, then, consists of exactly two components: a chain proposal, proposing tips for every chain as in the first bullet point above, and a reference (by hash) to the V-QC of an earlier view, the latter playing the role of the M-notarisation for its parent. A V-QC specifies, for each chain, a tip that is safe to extend, and thereby a total ordering, namely everything below the specified tips, in a fixed sweep order (Section 3.4). Validity requires the block’s chain proposal to extend, chain by chain, the tips specified by the block’s V-QC, just as a Minimmit proposal must extend its parent. The ordering therefore only ever grows. Figure 1 depicts the two layers as assembled by the consensus layer.

An L-QC for a block extends the ordering already specified by the V-QC the block references, and every subsequently finalised block extends it further. Since, as we shall see, the tips specified by any V-QC include the fresh tips of every honest chain (in the common case), whatever block is finalised next, its ordering already contains all honest chains’ blocks. This is the sense in which easy spoiling will have no analogue here. One distinction should be retained. A view v L-QC finalises tips chain by chain, but the portion of the ordering extractable from it *immediately* may be capped at the frontier of a lagging chain, with the remainder extracted a view later, via the view v V-QC referenced by the next leader block (even if the next leader attempts to censor transactions). In the good case the distinction all but disappears: every chain reaches its proposed tip in every vote, and a single L-QC orders everything, except that a block still in flight at the vote event, endorsed by some voters but not others, may defer the slots swept after its chain by a view (its own ordering was in any case a vote round away).

3.4 Proposal-relative voting

Throughout this subsection we describe a simplified version of the protocol, *without* extension votes; the next subsection adds them.

Chain proposals and positions. A leader block’s chain proposal contains, for each chain, an *anchor* (a certificate, or a tip already agreed safe) followed by references to up to d uncertified blocks. The referenced blocks are not themselves included. Each is specified by a single hash value, and the hash structure of blocks ensures, with no need for signatures within the proposal, that each referenced block is the child of the one before, so that a proposal names, per chain, a segment of chain extending the anchor. Nothing certifies that the referenced blocks *exist*, since a leader may reference blocks that were never produced; we shall see that this is harmless. (The reader may also worry that leader blocks grow large; we give calculations in Section 3.7 showing that they remain modest.) Voters respond with a vote reporting, for each chain, the greatest position up to which the voter has DA-voted the referenced blocks. A processor missing blocks on chain i votes a lower position on chain i and full positions elsewhere. No processor blocks or fetches on the critical path.

Two extraction rules. Let us now be precise about the two kinds of notarisation:

- An *L-notarisation* for a leader block is, exactly as in Minimmit, a set of $n - f$ votes for the block.
- A *V-notarisation* for view v is a set of $n - f$ view v messages, sent by distinct processors, each of which is either a vote for a view v leader block or a signed *novote* message, declaring that its sender did not vote in view v . At least $2f + 1$ of the votes must be for a single leader block, whose chain proposal the notarisation is said to *designate*.

(Correct processors that time out without voting send novote messages along with their nullify messages.⁴ Since a faulty leader may equivocate, we cannot insist that all votes be for the same block, but if some view v block receives an L-notarisation, one can check that every view v V-notarisation must designate *its* proposal. The requirement that a V-notarisation account for $n - f$ distinct processors, rather than comprising just $2f + 1$ votes, is what powers the quorum intersection arguments below.) As promised, the protocol actually manipulates succinct certificates standing for these sets, L-QCs and V-QCs respectively.

With votes now vectors of positions, each kind of notarisation specifies tips, extracted chain by chain, by rules differing only in a threshold:

- An L-notarisation finalises the leader block, and its *finalised tips* determine which transaction blocks are thereby finalised: on each chain, take the greatest proposed position endorsed by at least $3f + 1$ of the notarisation’s votes (that is, discard the top $3f$). Note that different L-notarisations for the same proposal may finalise different tips.
- A V-notarisation licenses view exit (in place of Minimmit’s M-notarisation), and its *safe-to-extend tips* are what the next leader must build on: on each chain, take the greatest proposed position endorsed by at least $f + 1$ of the notarisation’s votes for the designated proposal (that is, discard the top f).

The requirement relating the two rules is that, for any view v L-notarisation and any view v V-notarisation, the total ordering extracted from the V-notarisation must extend the ordering that

⁴The reader may wonder why novote messages are needed at all, given nullify messages. The distinction is that, as we shall see, a correct processor may sometimes vote and *later* also nullify (this is needed to ensure view progression), but a correct processor never both votes and sends a novote message for the same view. This exclusivity is what lets a V-notarisation account exactly for who did and did not vote, on which the extraction rules below rely.

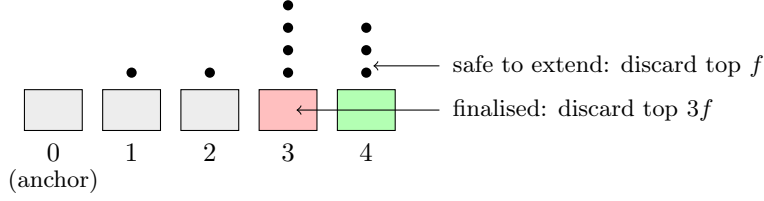


Figure 2: The two extraction rules on one chain, for $n = 11$, $f = 2$. The proposal makes four entries above the anchor, and the dots are the reported positions of the $n - f = 9$ votes of an L-notarisation, here 4, 4, 4, 3, 3, 3, 3, 2, 1. Discarding the top $3f = 6$ leaves position 3 as the finalised tip (red), while discarding the top $f = 2$ from any V-notarisation's tally containing these votes leaves position 4 as safe to extend (green). The safe-to-extend tip always sits at or above the finalised tip.

the L-notarisation finalises. This is what consistency needs. A processor may finalise via the L-notarisation, while the next leader builds on the safe-to-extend tips of whichever V-notarisation it happens to hold; were some V-notarisation's tips to fall short of a finalised tip, a later view could extend the chains past finalised blocks, and the logs of correct processors would fork. Quorum intersection delivers the requirement, for any $n \geq 5f + 1$. Behind any finalised tip stand at least $3f + 1$ votes, hence at least $2f + 1$ correct voters, of whom at most f are absent from any given V-notarisation. At least $f + 1$ of the V-notarisation's votes therefore sit at or above the finalised tip, so the greatest position endorsed by $f + 1$ of its votes, which is exactly what the safe-to-extend rule selects, is at or above the finalised tip, on every chain. Figure 2 shows the two rules at work on a single chain.

The total ordering extracted from a V-QC. It remains to specify how tips are turned into a log. We do this first for V-QCs, by a deterministic sweep. The ordering extracted from an L-QC, defined below, will then be an initial segment (typically all or most) of the ordering extracted from any same-view V-QC. To be concrete, consider a leader block for view v_2 , and let Q_1 be the V-QC it references, for some earlier view $v_1 < v_2$. As described above, Q_1 identifies the block's parent, and the block's chain proposal must extend the tips specified by Q_1 . Suppose now that view v_2 produces a V-QC Q_2 for this block. The ordering extracted from Q_2 extends the ordering already extracted from Q_1 , as follows. The new blocks on each chain (those lying above the tip specified by Q_1 and at or below the tip specified by Q_2) are appended by height relative to that chain's *previous* tip, interleaving the chains at equal offsets. Figure 3 gives an example with four chains, in which the red cells are the tips specified by Q_1 , the green cells those specified by Q_2 , and the sweep first appends every chain's first new block, then every chain's second, and so on, producing

$$1B, 2C, 3B, 4B, \quad 1C, 2D, 3C, 4C, \quad 1D.$$

Note that chain 2's block C is its *first* new block, and so is ordered in the first pass, alongside the other chains' first new blocks.

All honest tips are ordered. The crucial point is that the sweep orders *all* blocks below the tips specified by a V-QC, however little the leaders of the intervening views proposed for any given chain, and the tips specified by any V-QC include the proposed tip of every honest chain

D	1D	2D		
C	1C	2C	3C	4C
B	1B	2B	3B	4B
A	1A	2A	3A	4A
	1	2	3	4
	chains			

Figure 3: Extracting the total ordering. Columns are chains 1–4, and rows are heights A – D (A lowest). Red cells are the tips specified by Q_1 , and green cells the tips specified by Q_2 . The horizontal sweep appends the new blocks at equal offsets above the red tips, giving $1B, 2C, 3B, 4B$, then $1C, 2D, 3C, 4C$, then $1D$.

(in the common case). To see the latter, note that, on an honest chain whose proposed blocks reached all correct processors in time, *every* correct vote reports the full position, so among any $2f + 1$ votes of a V-QC’s tally at least $f + 1$ sit at the proposed tip, and the tip survives the discard of the top f . A faulty processor can lower the extracted tip only on its own chain, by withholding its own blocks.

The total ordering extracted from an L-QC. We now turn to L-QCs, and here we define the extracted ordering as follows. Let Q be an L-QC for a view v_2 leader block referencing the V-QC Q_1 . The ordering extracted from Q extends the ordering extracted from Q_1 by exactly the sweep above, performed towards Q ’s finalised tips, but subject to one stopping rule: the sweep halts at the first slot at which some chain has fallen short of its proposed tip.⁵ The stopping rule is needed because the sweep interleaves chains. A chain that has fallen short of its proposed tip may yet have blocks appear in that slot (via the view’s V-QC), and those blocks would precede, in sweep order, other chains’ blocks at later offsets. To see that this defines an initial segment of the ordering extracted from *any* view v_2 V-QC, recall the quorum intersection argument above, by which the finalised tips extracted from an L-QC sit at or below the safe-to-extend tips extracted from any same-view V-QC, chain by chain. In terms of Figure 3, any V-QC’s green frontier dominates any L-QC’s finalised frontier on every chain. Everything before the halting slot is thus ordered immediately, upon one round of voting, while the tail is ordered a view later, when the next leader block settles how far the lagging chains reached.

This also lets us make the guiding principle precise. A faulty non-leader processor cannot delay the *finalisation* of other chains’ blocks at all. What it can delay is their *placement* in the

⁵This form of the rule is exact for the simplified protocol of this subsection, where the proposed tip is a hard ceiling on a chain’s growth. Once votes may carry extensions (next subsection), the ceiling dissolves, since a chain *at* its proposed tip may yet grow, and the rule must be strengthened accordingly (see the definition of *settled* chains in Section 4). One might hope to also *relax* the rule, passing over a lagging chain when no vote of the L-QC endorses any block beyond the finalised tip. This is sound for the beyond-tip region, where growth requires $2f + 1$ endorsing votes, but not below the proposed tip, where a chain’s reported position advances at the threshold $f + 1$. There, f faulty votes and a single correct vote from outside the L-QC suffice, and no L-QC can certify that the latter does not exist.

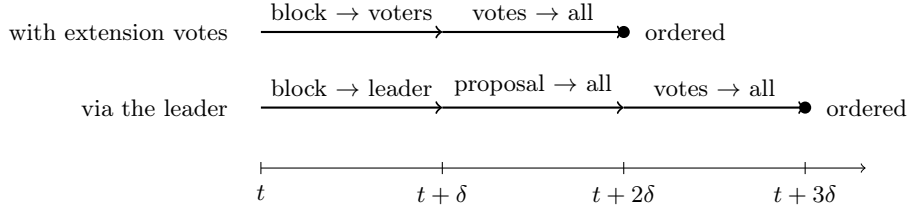


Figure 4: Best-case ordering of a transaction block disseminated at time t , with and without extension votes. Extensions remove the proposal leg, since the block need only reach the voters before they vote, not the leader before it proposes. (On average each route also incurs a wait for the next vote or proposal event, and the averages appear in Section 3.6.)

total ordering, at offsets beyond its own chain’s frontier, for roughly a view.⁶ This is the precise content of the qualifier ‘significantly’ in the guiding principle: other chains’ blocks are finalised immediately and totally ordered at worst a view later, while the faulty processor’s own chain is the only one whose finalisation it can hold up. Nor is the one-view wait costly by the standards of competing protocols, since, as we shall see in Section 3.6, it even completes sooner, in the common case, than Raptr’s first finalisation.

3.5 Extensions

We now add the second mechanism, extension voting. The reader should take two things from this subsection. First, extensions remove a message delay from the critical path, since a transaction block need only reach the *voters* before they vote, rather than the *leader* before it proposes. Second, they are what provides the block inclusion guarantee of Section 3.3, our form of censorship resistance. The mechanism itself is simple, and the one subtlety lies in how extension blocks enter the total ordering.

Racing the vote event. A block disseminated just before the leader formed its proposal would, so far, wait a full view before a leader can reference it. Multimitt instead lets votes carry *extensions*, meaning up to e fresh blocks (the *extension bound*, e.g. $e = 2$) that the voter has DA-voted above its reported position on each chain. Extensions are anchored at the voter’s reported position rather than at the proposed tip, which makes the frontier *leader-proof*: even if a leader’s entries for a chain are junk, correct voters report a lower position and attach the chain’s true recent blocks from there. The effect on latency is that, once a block has reached the voters, it is picked up by whichever votes are cast next, whatever proposal those votes respond to, and (in the good case) its transactions are ordered as soon as that single round of voting completes. Figure 4 depicts the leg this removes.

Ordering extension blocks. Recall how tips were extracted in the previous subsection. On each chain, we took the greatest proposed position endorsed by sufficiently many votes. That

⁶The one-view wait is robust, in that it does not rely on the goodwill of subsequent leaders. A valid proposal must anchor on a V-QC, and anchoring is all or nothing: the ordering extracted extends *everything* below the V-QC’s safe-to-extend tips, which carry every honest chain’s frontier. Even a Byzantine or censoring later leader therefore places the deferred blocks the moment it finalises anything at all; its only alternative is to finalise nothing and sacrifice its view, after which the same choice confronts the leader that follows.

rule is sound because the chain proposal names one specific block at each position, so that all votes on a chain refer to blocks along a single path, and ‘the greatest position endorsed by $3f + 1$ votes’ unambiguously identifies a block. Extension blocks enjoy no such pinning. They lie above anything the proposal names, and an equivocating producer may send incompatible blocks to different voters, so that two votes reaching the same height above the proposed tip may be endorsing incompatible blocks. Endorsement must then be counted branch by branch. The two extraction rules of the previous subsection extend accordingly, and the asymmetry between them now widens. It is important to keep the two cases separate.

- *Safe-to-extend tips (V-notarisations).* By the chain’s *position block* we mean the block selected by the rank rule in Section 3.4 above: the block at the greatest proposed position that at least $f + 1$ of the notarisation’s votes endorse. With extension votes incorporated, the safe-to-extend tip becomes the deepest block, extending the chain’s position block, that at least $2f + 1$ of the notarisation’s votes endorse (the position block standing if there is none). This carry needs no further condition, and extraction from V-QCs remains exactly as robust as before. In the common case an honest chain’s fresh blocks are endorsed by every correct vote, hence by at least $2f + 1$ votes of any V-notarisation’s tally, and so enter its safe-to-extend tips *whatever the leader proposed* — junk entries included, since extensions are anchored at voters’ reported positions. This rule is what keeps the frontier leader-proof.
- *Finalised tips (L-notarisations).* A place in the total order requires much more, and at $n = 5f + 1$ the requirements are forced (Section 5 proves this, and determines the exact threshold for every n): *all* $n - f$ votes of the L-QC must endorse the extension block, and the chain’s finalised position must be the proposed tip itself, so that the extension sits above a fully endorsed proposal path. Note, however, what unanimity does and does not require. The votes need not agree on which extension blocks exist: each of the $n - f$ votes must merely *endorse* the block in question, by reporting an endorsed tip at or beyond it, and different votes may endorse different continuations above it. Nor is the quorum fixed, since any $n - f$ votes form an L-QC, and a processor may select among the votes it has received (see ‘Unanimity is less demanding than it appears’ below).

The message to carry forward is the positive one. If a transaction block produced by a correct processor reaches every correct processor before it votes, and lies within ϵ of the previous tips, then every correct vote endorses it, and the pool of correct votes finalises it within the view, whatever the f faulty voters do. (Placement in the total order follows at once, except when the sweep queues the block behind a lagging chain’s slot, in which case it completes via the next finalised leader block.) This requires the chain’s proposed entries to be genuine, which is automatic under a correct leader; junk entries are a Byzantine leader’s one lever here, and Sections 3.6 and 5 quantify what survives them: the block still gathers at least $n - 2f$ endorsing votes in *every* L-QC of the view, however the leader and the faulty voters behave, which settles its membership of the eventual ledger and orders it at latest via the next finalised leader block (the block inclusion theorem).

Unanimity is less demanding than it appears. First, observe that the finality requirement for L-notarisations is no stronger in kind than the quorum requirement of any BFT protocol. In an execution with f faulty processors, a quorum of $n - f$ consists of exactly the correct processors, so PBFT, too, finalises its leader’s block only when *every* correct processor votes for it. The

extension rule asks for the same thing per transaction block rather than per leader proposal: the block must have reached the voters. Three further observations soften the requirement.

- What is required is unanimity within a *single L-QC*, not across the network, since an L-QC is any $n - f$ votes for the block, and a processor extracting the ordering may consider whichever such sets its received votes provide.
- In the common case, the $n - f$ *correct* votes agree on every honest chain's fresh blocks, each of which reached all correct processors before they voted, so an L-QC with the required unanimity is always available, whatever the f faulty voters do.
- No search through candidate L-QCs is needed. As we show in Section 4, *everything finalised or ordered by any assemblable L-QC can be read off directly from the pool of received votes*, chain by chain (for extensions, the relevant quantity is simply the deepest block on the chain that $n - f$ of the received votes endorse), with the pool's conclusions always at least as strong as, and compatible with, those of every assemblable L-QC. This observation is arguably what makes unanimity workable as a threshold: the protocol never asks whether some particular certificate achieves unanimity, but only what the received votes, taken together, endorse. Since the orderings extracted from different L-QCs for the same view are always compatible, a processor may also safely emit more as further votes arrive.

3.6 What is achieved, and the comparison with Raptr

With the mechanisms now in place, we collect what the protocol guarantees, case by case, making the comparison with Raptr quantitative along the way. Precise statements and proofs appear in Section 5. Recall from Section 3.3 the distinction between the blocks a QC finalises and the portion of the total ordering extractable from it immediately. Two guarantees hold unconditionally in all cases below: consistency, and the damage localisation of Section 3.4. We say a Byzantine leader is *greedy* if it seeks the finalisation of its leader block (say, for block rewards) and avoids leaving evidence of provable misbehaviour (say, to avoid slashing). None of our results require this assumption. Greedy leaders are arguably the realistic censorship adversary, however, and the guarantees against them are significantly stronger than those against an adversary with nothing to lose.

- *The good case* (synchrony, all processors correct). Every view finalises, and a single L-QC orders *everything* endorsed by all correct voters, fresh extensions included. Concretely, consider any transaction block disseminated at least δ before the first correct processor votes and lying within ϵ of the proposed tips. Every vote endorses the block, and the view's L-QC places it in the total ordering. A block still in flight at the vote event waits one view. A transaction block is thus ordered 3δ after dissemination on average, and 2δ at best, which is optimal.⁷ Consensus objects total tens of kilobytes per view (Section 3.7), independent of transaction volume. Now for the comparison. In Raptr, a batch disseminated at time t reaches the leader by $t + \delta$, and waits on average δ for inclusion in the next leader block, leader blocks being produced every 2δ . The batch is ordered after that block is sent and two rounds of voting complete, at $t + 5\delta$ on average. In Multimitt, the transaction block reaches all voters by $t + \delta$, and waits on average δ for the next vote event. A single round of

⁷As in Section 1, latency is measured from the dissemination of the transaction block itself; measured from the leader's proposal, the figure would simply be 2δ .

voting then completes the ordering, giving the figures above. Of the 2δ saved on average, one δ is bought by the assumption $5f + 1 \leq n$ (one round of voting rather than two), and one by extension votes (the block need not travel to the leader and back out inside a proposal).

- *The common case* (synchrony, correct leader, up to f other processors faulty). From its pool of received votes, in the view itself, v say, every correct processor finalises each honest chain through its proposed tip and the extensions endorsed by every correct voter, whatever the f faulty voters do. That is, an honest chain finalises everything that reached the correct voters in time, up to e blocks beyond the proposed tips. A faulty chain finalises as far as $3f + 1$ votes endorse. The sweep that assembles the finalised tips into the total ordering may halt at a lagging faulty chain's slot. In that bad case, the deferred tail is then ordered, via view v 's V-QC, by the next view that finalises a block, immediately if the next leader is correct or merely greedy. An honest chain's block disseminated at time t is thus finalised by $t + 3\delta$ on average. The block is placed in the total ordering at that same moment or, if the sweep queues the block behind a lagging faulty chain's slot, at most 2δ later, with the next view's finalisation. Raptr's figures, in contrast, degrade twice over. First, its views (which the Raptr paper calls rounds) stretch from 2δ to 3δ . The early view exit requires a quorum certificate on the *full* prefix of the proposal, and so a quorum of voters each holding every referenced batch. A single withholding producer per view ensures that no processor other than the leader can be such a voter, and views then advance only upon the certificate formed by the second round of voting. With leader blocks every 3δ , the average wait for inclusion grows from δ to 1.5δ . For a batch disseminated at time t , a proposal including the batch is finalised at $t + 5.5\delta$ on average. Finalised here means only that the *proposal* commits. Whether the batch itself is thereby ordered is the second degradation: by easy spoiling, the committed prefix may stop short of the batch, ordering little or nothing beyond included certificates. Multimmit's placement guarantee, by contrast, holds however the next leader behaves. The next leader block must anchor on a view v V-QC, whose safe-to-extend tips carry every honest chain's frontier. If that block is finalised at all, Byzantine or not, it places the transaction blocks in question in the total ordering. Easy spoiling has no analogue, since f faulty producers delay only their own chains' contributions. In summary, Multimmit achieves a total ordering that faulty producers cannot influence *before* Raptr achieves the ordering that they can.
- *A greedy Byzantine leader*. The two conditions in the definition of greed give two corresponding guarantees. Because the leader seeks finalisation, the block inclusion theorem applies: *either the view finalises nothing, or every fresh honest block that reached the voters is membership-finalised in the view* (no competing block ever certifiable) *and ordered at latest by the next finalised leader block*. A greedy leader is therefore unable to finalise its block while censoring. The guarantee of ordering does not depend on the behaviour of subsequent leaders, since *any* later finalised leader block orders the blocks in question. A greedy leader finalises during synchrony, so the wait is a single view. Because the leader also avoids leaving attributable evidence, junk proposal entries are unavailable, as is equivocation. (Junk entries are the *leader's* one lever against an honest chain's in-view order finality, and are unanswerable under a challenge to exhibit the signed headers; the accountability paragraph of Section 5 describes which misbehaviours are provable and which merely attributable.) The leader's remaining option is omission, i.e. proposing less than it received,

which is indistinguishable from slowness. Extension votes bound the resulting damage. Blocks within the extension bound e of the previous tips are finalised in-view whatever the leader proposes, and placed in the total ordering at latest via the next finalised leader block; only the excess on under-proposed chains waits for the next view.

- *An arbitrary Byzantine leader.* A leader with nothing to lose can suppress its entire view, in which case correct processors nullify within bounded time, nothing is finalised by anyone, and the view is skipped. (The backup-proposal extension of Section 6 ensures that silence no longer suffices for this, whenever the view’s backup is correct, at the price of one further proposal per view.)

3.7 Sizes: some concrete numbers

We have promised at several points that the objects the protocol manipulates remain small. Here are rough numbers. Suppose hashes are 32 bytes, signatures and signature shares (e.g. BLS) are 48 bytes, and take $d = 3$ and $e = 2$. We consider $n = 100$ validators, and, recalling from the start of this section that the number of block producers may differ from the number of validators, we let K denote the number of chains and consider both $K = 100$ and $K = 40$ (the latter being representative of intended applications). Per-chain vectors scale with K , while quorum-related components scale with n .

- *DA-certificates* are constant size: a block header (≈ 75 bytes) plus one threshold signature, so ≈ 125 bytes.
- *Chain proposals* contain, per chain, an anchor (at worst a DA-certificate) and up to d payload hashes: under 230 bytes per chain, so $\approx 23\text{KB}$ at $K = 100$ and $\approx 9\text{KB}$ at $K = 40$.
- *Votes* contain a view number, a block hash and one signature (≈ 90 bytes), plus a position per chain (2 bits each), plus extensions: at most e hashes per chain, and only for chains on which the voter holds blocks beyond the proposal. Even with every chain extended by e blocks, this is under 7KB at $K = 100$ and under 3KB at $K = 40$; in light load, votes are a few hundred bytes.
- *L-QCs and V-QCs*, in the good case, comprise the proposal, one reference extension vector, a voter bitmap ($n/8 \approx 13$ bytes) and a *single* aggregate signature: $\approx 30\text{KB}$ at $K = 100$ and $\approx 12\text{KB}$ at $K = 40$. Outside the good case, each deviating vote adds its differences from the reference. In the worst case, with every quorum vote deviating from every other, QCs grow to roughly 0.5MB at $K = 100$ and 0.2MB at $K = 40$ (the price of per-chain vote vectors); f Byzantine voters alone cause at most a quarter of that, benign disagreement concentrates on a few chains, and the aggregate signature remains a single group element throughout.
- *Leader blocks* comprise a hash reference to a V-QC and a fresh proposal: $\approx 23\text{KB}$ at $K = 100$ and $\approx 9\text{KB}$ at $K = 40$. (The referenced V-QC is not included, travelling separately.)

Two points deserve emphasis.

- All of these sizes are independent of transaction volume: the consensus layer never carries transaction data, which travels only once, from each producer to all, in the chain layer.

- The totals are modest in comparison with what they coordinate. Precisely because transaction data travels only once, the protocol can (modulo execution costs) finalise transactions at roughly the rate at which validators can receive data. With views lasting around 2δ , say 100–200ms, a single view at line rate then orders tens of megabytes of transactions on a commodity gigabit link (125MB/s), and hundreds of megabytes at ten gigabits, against consensus objects totalling tens of kilobytes either way.

4 The formal specification

In what follows, we suppose that all messages are signed by the sender. We say ‘disseminate’ to mean ‘send to all processors’. When a correct processor is instructed to send a message to itself, it regards that message as immediately received. The protocol is parameterised by two small constants, discussed in Section 3: the *pipelining depth* $d \in \mathbb{N}_{\geq 1}$ and the *extension bound* $e \in \mathbb{N}_{\geq 0}$. For concreteness, one may take $d = 3$ and $e = 2$.

The pseudocode uses a number of message types, local variables, functions and procedures, detailed below. Message types and local variables are summarised in Tables 1 and 2, and the functions extracting tips and orderings in Table 3; the pseudocode appears in Algorithms 1 and 2.

The local variable $\mathcal{S}$. This variable is maintained locally by each processor p_i and stores all messages received. It is considered to be automatically updated, i.e., we do not give explicit instructions in the pseudocode updating $\mathcal{S}$. Several of the objects defined below (DA-certificates, nullifications, and QCs) are formed from sets of constituent messages: any processor may form such an object upon receiving its constituents. Accordingly, $\mathcal{S}$ is regarded as containing such an object whenever it contains an object of which it is an entry, or messages from which it may be formed; this too happens automatically, with no explicit instructions in the pseudocode. Initially, $\mathcal{S}$ contains only the ‘genesis objects’, defined below: the genesis blocks $b_{\text{gen}}^1, \dots, b_{\text{gen}}^n$ and a DA-certificate for each, and the genesis leader block ℓ_{gen} together with the V-QC Q_{gen} and an L-QC for ℓ_{gen} .

4.1 The chain layer

Transaction blocks. For each $i \in \{1, \dots, n\}$, the *genesis block of chain i* is the tuple $b_{\text{gen}}^i := (i, 0, \lambda, \lambda)$, where λ denotes the empty sequence. A transaction block other than a genesis block is a tuple

$$b = (i, h, x, \text{Tr}),$$

signed by p_i , where:

- $i \in \{1, \dots, n\}$ (thought of as the chain to which b belongs);
- $h \in \mathbb{N}_{\geq 1}$ (thought of as the height of b);
- x is a hash value (used to specify b ’s parent), and;
- Tr is a sequence of distinct transactions.

We write $b.\text{chain}$, $b.\text{height}$, $b.\text{par}$ and $b.\text{Tr}$ to denote the corresponding entries of b . The *header* of b is $\text{hd}(b) := (i, h, x, H(\text{Tr}))$; the required signature is p_i ’s signature on $\text{hd}(b)$, and we define

$H(b) := H(\text{hd}(b))$.⁸ The *parent* of b is the block whose hash is $b.\text{par}$. The parent of a block of height h is required to be of height $h - 1$ on the same chain. The *ancestors* of b are b and all ancestors of its parent (a genesis block has only itself as ancestor). Two transaction blocks on the same chain are *incompatible* if neither is an ancestor of the other. A block *extends* another if the latter is an ancestor of the former.

DA-votes. A *DA-vote* (‘data availability vote’) for the transaction block b by p_j is a message $(\text{da}, \text{hd}(b), \rho)$, where ρ is p_j ’s $(n - 2f)$ -threshold share on $(\text{da}, \text{hd}(b))$. (A DA-vote is thus self-describing, specifying the chain, height, and hash of the block it votes for.)

DA-certificates. A *DA-certificate* for the transaction block b is a pair $(\text{hd}(b), \tau)$, where τ is the $(n - 2f)$ -threshold signature on $(\text{da}, \text{hd}(b))$, formed from the shares of $n - 2f$ DA-votes for b by different processors. A DA-certificate is thus of constant size, and specifies the chain, height, parent, and hash of the block it certifies, and is interpretable (and verifiable) by processors that have not received b itself. If c is a DA-certificate for b , we write $c.\text{chain}$, $c.\text{height}$ and $c.\text{hash}$ to denote $b.\text{chain}$, $b.\text{height}$ and $H(b)$ respectively. Each genesis block b_{gen}^i is considered DA-certified at the start of the protocol execution.

When p_j may DA-vote for b . Suppose p_j has received a correctly signed transaction block $b = (i, h, x, \text{Tr})$.⁹ Let h' be the greatest height $< h$ such that $\mathcal{S}$ (as locally defined for p_j) contains a DA-certificate c for some block of height h' on chain i . Then p_j *may DA-vote for b* if:

- (i) p_j has not previously sent a DA-vote for any block of height h on chain i ;
- (ii) $h - h' \leq d$;
- (iii) $\mathcal{S}$ contains blocks $b_{h'+1}, \dots, b_h = b$ on chain i , with $b_{h'+1}.\text{par} = c.\text{hash}$ and $b_{h''}.\text{par} = H(b_{h''-1})$ for each $h'' \in (h' + 1, h]$, and p_j has sent a DA-vote for $b_{h''}$ for each $h'' \in (h', h)$.

Condition (i) ensures that correct processors send at most one DA-vote per (chain, height) pair; line 4 of Algorithm 1 is understood to process eligible blocks in increasing order of height, each DA-vote sent counting as already sent for the conditions above within the same timeslot, so that a processor receiving a path of new blocks DA-votes them in sequence. Together with condition (iii), condition (i) ensures that all DA-certified blocks for a given chain are compatible with each other (Lemma 1). Condition (ii) bounds the number of uncertified blocks at the head of any chain. DA-votes for b are sent to $p_{b.\text{chain}}$ (rather than disseminated), who forms and disseminates the DA-certificate.

The predicate $\text{Ready}(\mathcal{S})$. This predicate is used by p_i to decide whether to produce the next block on its own chain. In what follows, b denotes the greatest block p_i has produced on chain i (or b_{gen}^i), h^c denotes the greatest height at which $\mathcal{S}$ contains a DA-certificate for chain i , and $\text{pending}(\mathcal{S})$ denotes the set of transactions received by p_i and not included in b or any of its ancestors. The predicate $\text{Ready}(\mathcal{S})$ holds if:

⁸The signature is thus verifiable, and equivocation provable, from headers alone: two blocks signed by p_i at the same height, or a signed block at height $h + 1$ whose x differs from the hash of p_i ’s signed block at height h , constitute constant-size evidence of misbehaviour. Note that the signature is excluded from $H(b)$, so that the identifier of a block is a function of its content only. In implementation, $H(\text{Tr})$ may be taken to be the root of a Merkle tree over Tr .

⁹By any route: since blocks are signed by their producers, a block received by relay is as good as one received directly.

- (i) (*window*) $b.\text{height} + 1 - h^c \leq d$;
- (ii) (*work*) $\text{pending}(\mathcal{S}) \neq \emptyset$, and;
- (iii) (*pacing*) $|\text{pending}(\mathcal{S})| \geq B$, or at least θ time has passed since p_i produced b ,

where the *batch bound* $B \in \mathbb{N}_{\geq 1}$ and the *production interval* $\theta \geq 0$ are further protocol parameters. Condition (i) is required for the analysis, while conditions (ii) and (iii) are policy, and the analysis requires of them only that, whenever $\text{pending}(\mathcal{S})$ remains non-empty, the pacing condition is satisfied within bounded time.¹⁰

The procedure $\text{ProduceNext}()$. This procedure is executed by p_i to extend its own chain, and may be executed only when $\text{Ready}(\mathcal{S})$ holds. Recall from the definition of $\text{Ready}(\mathcal{S})$ that b denotes the greatest block p_i has produced on chain i (or b_{gen}^i). To execute the procedure, p_i :

- Forms a sequence Tr of distinct transactions comprising all of $\text{pending}(\mathcal{S})$,¹¹ and;
- Disseminates the block $(i, b.\text{height} + 1, H(b), \text{Tr})$.

4.2 The consensus layer

The function $\text{lead}(v)$. The value $\text{lead}(v)$ specifies the leader for view v . To be concrete, we set $\text{lead}(v) := p_{j+1}$, where $j = v \bmod n$.

Chain proposals. A *chain proposal* is a function C mapping each $i \in \{1, \dots, n\}$ to a pair $C(i) = (a_i, x_i)$, where:

- a_i (the *anchor* for chain i) is either a pair (y, h) consisting of a hash value y and a height h , or a DA-certificate for chain i , and;
- $x_i = (x_i^1, \dots, x_i^{m_i})$ is a possibly empty sequence of $m_i \leq d$ hash values, called *payloads* (thought of as the values $H(\text{Tr})$ of a sequence of consecutive blocks on chain i , the first extending the anchor).

The *base* for chain i is the block specified by the anchor. Note that the anchor always specifies the base's hash and height explicitly, so that a chain proposal is interpretable in isolation.¹² For $0 \leq k \leq m_i$, we write $\text{Prop}(C, i, k)$ for the block specified at *position* k on chain i : position 0 is the

¹⁰Setting $B = 1$ and $\theta = 0$ gives eager production: minimal inclusion latency, but under low load each straggling transaction costs a block of its own, namely a header, a signature, a round of DA-voting, and a slot of the d -window. Larger B (naturally denominated in bytes rather than transaction count) amortises this per-block overhead under high load, while θ bounds the delay a lone transaction can suffer under low load. The familiar ‘batch full or timer expired’ policy is exactly (iii). Under sustained load, production is in any case paced by the window: at most d blocks per certificate round-trip. Values of θ commensurate with the view cadence are natural, since a chain contributes at most d new blocks to any one leader block.

¹¹Any inclusion policy would do, provided every transaction entering $\text{pending}(\mathcal{S})$ is included by some later execution of ProduceNext . Together with the requirement, stated with $\text{Ready}(\mathcal{S})$, that the pacing condition be satisfied within bounded time while transactions are pending, this is exactly what Step 1 of the proof of Theorem 2 uses.

¹²This is deliberate. The alternative, in which an anchor defaults to the corresponding coordinate of $\text{Tips}(Q)$ for the accompanying V-QC Q , would make the value Tips (defined below) a recursion over the entire history of V-QCs: position-0 votes would resolve to a block identified only by the *previous* V-QC, and so on back to genesis. Explicit anchors keep every V-QC interpretable in isolation, keep proposal validity locally checkable, and mean that a single leader block tells a recovering processor exactly which blocks to fetch on every chain.

base and, recursively, position $k \geq 1$ is the block with header $(i, h_0 + k, H(\text{Prop}(C, i, k-1)), x_i^k)$, where h_0 is the base's height. That is, entries specify blocks by payload alone, with chain, height and parent supplied by context. The specified blocks thus form a chain *by construction*, and the producer's signature on any of them is verifiable against the reconstructed header.¹³ The *proposed tip* for chain i is $\text{Prop}(C, i, m_i)$; when $m_i = 0$, as is natural for an idle chain, the proposed tip is the base itself, the proposal making no new entries for the chain. Whether an anchor of the form (y, h) is *correct* (names the tip specified by the accompanying V-QC) is a validity condition on leader blocks, defined below.

Leader blocks. The *genesis leader block* is the tuple $\ell_{\text{gen}} := (0, \lambda, \lambda)$. A leader block other than ℓ_{gen} is a tuple

$$\ell = (v, q, C),$$

signed by $\text{lead}(v)$, where $v \in \mathbb{N}_{\geq 1}$ (thought of as the view corresponding to ℓ), q is a hash value, and C is a chain proposal. The value q is required to be the hash $H(Q)$ of a V-QC Q (defined below) for an earlier view. We then say that ℓ *references* Q , which is thought of as identifying ℓ 's parent, and write $\ell = (v, H(Q), C)$. Note that the V-QC itself is not included in the block. V-QCs are forwarded to all upon first receipt, the leader's anchor among them (see **SelectAnchor** below), so processors obtain them separately, and leader blocks stay small.¹⁴ We write $\ell.\text{view}$, $\ell.q$ and $\ell.C$ for the corresponding entries, and define $H(\ell) := H(v, q, H(C))$. As for transaction blocks, the leader's signature is excluded from $H(\ell)$, so possession of the tuple (v, q, C) itself thus determines $H(\ell)$, and so suffices to verify signatures on votes for ℓ (defined below). The QCs below accordingly include leader blocks directly (without the leader's signature, which they do not need).

Votes. A *vote* for the leader block $\ell = (v, H(Q), C)$ by p_j is a message of the form

$$(\text{vote}, v, H(\ell), \pi, \varepsilon, s),$$

where π maps each chain i to a position $\pi(i) \in \{0, \dots, m_i\}$, ε maps each chain i to a sequence $\varepsilon(i)$ of at most e payloads, and s is p_j 's signature on $(\text{vote}, v, H(\ell), \pi, \varepsilon)$. Since the single signature covers the entire content, no party can exhibit a vote while withholding or altering any part of (π, ε) .¹⁵ The intended semantics, enforced for correct voters by the voting rules below: p_j has DA-voted for $\text{Prop}(C, i, k)$ for every $k \in [1, \pi(i)]$, and $\varepsilon(i)$ specifies (by payloads, with chain, heights and parents supplied by context, as for proposal entries) a sequence of consecutive blocks on chain i extending $\text{Prop}(C, i, \pi(i))$, each of which p_j has DA-voted for. We refer to ε as the

¹³Payload entries do not certify that the specified blocks exist. A leader may propose junk payloads, in which case (correct) voters simply report lower positions, with extensions (below) anchored at their reported positions. Junk entries thus capture neither votes nor extensions, though they do render the chain unsettled (in the sense defined under 'The total ordering' below), deferring emission of the slots the sweep visits after it by a view. A Byzantine leader stalls part of its own view's emission, nothing more.

¹⁴This also suits implementation: the wire format of leader blocks matches Minimmit's, with the parent reference a single hash value.

¹⁵For the same reason, the QCs below never *abridge* a vote: although they record votes succinctly (most votes are recorded implicitly, as 'standard'), the record always determines each constituent vote's full (π, ε) . This is important for the extraction rules below: were votes to carry a separately verifiable 'participated in view v ' component, usable in a QC without the vote's content, the assembler of a V-QC could conceal the positions of correct voters, and the quorum-intersection argument underlying **Tips** would fail.

vote's *extensions*; note that extensions are anchored at the vote's reported position, not at the proposed tip.¹⁶

The *endorsed tip* of a vote, for chain i , is the last block specified by $\varepsilon(i)$ if $\varepsilon(i) \neq \lambda$, and $\text{Prop}(C, i, \pi(i))$ otherwise. We say a vote *endorses* a transaction block b if b is an ancestor of, or equal to, the vote's endorsed tip on b 's chain. (Thus a vote endorses every block on the path it endorses, however that path is split between positions and extensions.) The extraction rules below are stated in terms of this relation.

Standard votes, deviations, and tallies. Fix a leader block ℓ and a *reference extension vector* E^* (a value of the same type as ε). Recalling that m_i denotes the number of payload entries that ℓ 's chain proposal makes for chain i (so that $\pi(i) = m_i$ expresses full endorsement of the proposed tip), a vote for ℓ is *standard* (w.r.t. E^*) if $\pi(i) = m_i$ for every i and $\varepsilon = E^*$. The *deviation record* of a vote is the list of pairs $(i, \pi(i))$ for those chains with $\pi(i) < m_i$, together with ε if $\varepsilon \neq E^*$. A *tally* for ℓ is a tuple $T = (E^*, P, D)$, where P is a set of processor indices and D assigns deviation records to some subset of P . The tally describes one vote for ℓ per processor in P : standard w.r.t. E^* for processors outside the domain of D , and otherwise as given by D . The QCs below consist of a leader block, a tally, and an aggregate signature (plus, for V-QCs, records of the quorum members not voting for ℓ). In the good case (synchrony, all processors correct, no block still in flight at the vote event) all votes are standard w.r.t. a single E^* , so D is empty and all constituent signatures are on a single common message: verification requires one pairing computation. Each deviating vote adds its deviation record and one message to the aggregate, and the worst case is bounded by the quorum size.¹⁷

The functions $\text{Positions}(\mathcal{S}, \ell)$ and $\text{Extensions}(\mathcal{S}, \ell)$. Used by p_j to form its vote for $\ell = (v, H(Q), C)$. $\text{Positions}(\mathcal{S}, \ell)$ maps each i to the greatest $k \leq m_i$ such that p_j has DA-voted for $\text{Prop}(C, i, k')$ for all $k' \in [1, k]$. $\text{Extensions}(\mathcal{S}, \ell)$ maps each i to the longest sequence of payloads $(H(b^1.\text{Tr}), \dots, H(b^m.\text{Tr}))$ with $m \leq e$ such that $b^1, \dots, b^m$ are blocks on chain i that p_j has DA-voted for, with b^1 's parent the block $\text{Prop}(C, i, \text{Positions}(\mathcal{S}, \ell)(i))$ and b^k the parent of b^{k+1} for each $k < m$ (and to λ if there is no such sequence).

Novote and nullify messages. For $v \in \mathbb{N}_{\geq 1}$, a *novote*(v) message by p_j is a message (novote, v, s) with s p_j 's ordinary signature on (novote, v) , and a *nullify*(v) message by p_j is a message $(\text{nullify}, v, \rho)$ with ρ p_j 's $(2f + 1)$ -threshold share on $(\text{nullify}, v)$. A correct processor that times out in view v without voting sends both, while a correct processor that nullifies view v *after* voting (see Algorithm 2, lines 16–19) sends only the latter. Correct processors never send both a vote for a view v leader block and a novote(v) message.¹⁸

¹⁶This is what makes the frontier leader-proof: if the leader's entries for chain i are junk or stale, correct voters report a lower position and attach the chain's true recent blocks as extensions from there.

¹⁷Deviation records may in turn be compressed, e.g. by recording only the coordinates at which a deviating extension vector differs from E^* , and by sharing identical records across voters; we do not specify this further. A Raptor-style scheme of per-position keys, under which position deviations would not fragment the aggregate, does not pay here: extensions cannot be keyed, ranging as they do over arbitrary hash values, and extensions concern precisely the blocks racing the vote event, which is where benign disagreement concentrates.

¹⁸This property is what separates V-QCs (which certify how far each chain may safely be extended) from mere view progression, and the analysis relies on it. It is the reason novote and nullify are distinct message types, and the reason the former is an ordinary signature (novotes appear individually attributed inside V-QCs) while the latter is a threshold share (nullifications need certify nothing about who nullified).

Nullifications. A *nullification* for view v is a message $(\text{nullification}, v, \tau)$, where τ is the $(2f + 1)$ -threshold signature on $(\text{nullify}, v)$, formed from the shares of $\text{nullify}(v)$ messages by $2f + 1$ different processors. (A nullification is thus of constant size, and self-describing: it specifies the view it nullifies and what τ is a signature on. The tag also syntactically distinguishes nullifications from $\text{nullify}(v)$ messages, whose shares are group elements of the same kind as τ .)

L-QCs. An *L-QC* for the leader block ℓ is a tuple

$$(\ell, T, s),$$

where $T = (E^*, P, D)$ is a tally for ℓ with $|P| = n - f$, and s is the aggregate of the signatures of the $n - f$ votes described by T . An L-QC thus certifies a set of $n - f$ votes for ℓ by distinct processors, and we identify it with that set.

V-QCs. A *V-QC* for view v is a tuple

$$Q = (\ell, T, N, X, s),$$

certifying a set W of $n - f$ view v messages by distinct processors, where, writing C for the chain proposal of ℓ :

- $T = (E^*, P, D)$ is a tally for ℓ with $|P| \geq 2f + 1$; the votes it describes are referred to as the votes *for* C in W ;
- N is a set of processor indices, each contributing a $\text{novote}(v)$ message to W ;
- X assigns, to some further set of processor indices, records determining complete votes $(v, H(\ell'), \pi, \varepsilon)$ for other view v leader blocks ℓ' ,¹⁹ and;
- s is the aggregate of the signatures of all messages in W (the novotes contributing a single common message to the aggregate).

The sets P and N and the domain of X are disjoint, with sizes summing to $n - f$. We write $Q.\text{view}$ for v , and identify Q with the pair (W, C) . There is a distinguished initial V-QC Q_{gen} for view 0, considered received by all processors at the start of the execution, with $\text{Tips}(Q_{\text{gen}})_i := b_{\text{gen}}^i$ for each i .

The function $\text{Tips}(Q)$. Defined for any V-QC $Q = (W, C)$, the value $\text{Tips}(Q)$ maps each chain i to a transaction block, and is thought of as specifying, for each chain, a tip that is *safe to extend* given Q . For each chain i , let V_i be the votes for C in W (so $|V_i| \geq 2f + 1$), and proceed as follows.

- *Positions.* Order the values $\{\pi(i) : \pi \in V_i\}$ from greatest to least (with multiplicity), discard the first f , and let k_i be the next value. Set $b_i := \text{Prop}(C, i, k_i)$.

¹⁹Votes for other leader blocks can exist only if $\text{lead}(v)$ equivocates, in which case any two such blocks, being signed, are evidence of misbehaviour. Note that the records in X determine such votes in full: per the atomicity footnote above, a QC's record of a vote (however succinctly encoded) must always determine the vote's complete content.

- *Extension carry.* Redefine b_i to be a deepest block on chain i that extends b_i and that at least $2f + 1$ votes in V_i endorse, if there is one, choosing deterministically (say, by least hash) if there is more than one.²⁰

Then $\text{Tips}(Q)_i := b_i$. Since anchors specify bases explicitly, each $\text{Tips}(Q)_i$ is given as an explicit (hash, height) pair by Q alone: no other V-QC is needed to interpret it. Note also that each coordinate of $\text{Tips}(Q)$ has been DA-voted by at least one correct processor (at least $f + 1$ when the extension carry applies), which therefore possesses the block and all of its ancestors down to a DA-certified block. This secures data availability for all blocks entering the total ordering defined below, quantified precisely by the availability lemma of Section 5.

The function $\text{Tips}^*(Q)$. Defined for any L-QC Q for the leader block $\ell = (v, H(Q'), C)$, the value $\text{Tips}^*(Q)$ maps each chain i to a transaction block, and is thought of as specifying, for each chain, the tip *finalised* by Q . For each chain i , proceed as follows.

- *Positions.* Order the values $\{\pi(i)\}$ over the $n - f$ votes in Q from greatest to least (with multiplicity), discard the first $3f$, and let k_i^* be the next value.
- *Extensions.* If $k_i^* = m_i$, then b_i^* is the deepest block on chain i extending $\text{Prop}(C, i, m_i)$ that *every* vote in Q endorses (or $\text{Prop}(C, i, m_i)$ itself if there is none). If $k_i^* < m_i$, then $b_i^* := \text{Prop}(C, i, k_i^*)$.

Then $\text{Tips}^*(Q)_i := b_i^*$. Note the asymmetry of thresholds: positions use the rank rules (discard f , respectively $3f$), while extensions use endorsement thresholds ($2f + 1$, respectively unanimity above a fully endorsed proposal path). One might hope to treat both by a single rank rule, counting extension depths along with positions. The rank rules are sound precisely because the proposal pins one block per position, so votes on a chain's positions lie on a common path and a rank determines a block. Extensions lie above anything pinned. Under an equivocating producer, votes may split between incompatible branches, a rank then determines only a height, and the branch-aware repair at the same thresholds is impossible; whence the strengthened thresholds, which are forced at $n = 5f + 1$.²¹

²⁰For $n = 5f + 1$ the choice never arises: two incompatible candidates would each require $2f + 1$ endorsing votes, no vote endorses both, and $|V_i| \leq n - f = 4f + 1$. For larger n it may arise, but any deterministic choice is safe. To see this, suppose an extension block t on chain i is finalised by some same-view L-QC: all $n - f$ of that L-QC's votes endorse t , so at least $n - 2f$ correct processors do, of which at least $n - 3f \geq 2f + 1$ appear in V_i ; meanwhile any block incompatible with t has at most f correct endorsers, so musters at most $2f$ votes. Every candidate here is thus compatible with t , and every *deepest* candidate (being at least as deep as t , which itself qualifies) extends t .

²¹Quorum intersection gives that $\text{Tips}(Q'')_i$ extends $\text{Tips}^*(Q)_i$ for every V-QC Q'' of the same view (Lemma 7). For positions: at least $3f + 1$ of Q 's votes report positions $\geq k_i^*$, so at least $f + 1$ of them, from correct voters, appear in the tally of Q'' , whence Q'' 's rank is at least k_i^* . For extensions: $k_i^* = m_i$ pins Q'' 's rank at m_i exactly, and at least $n - 3f \geq 2f + 1$ of the votes endorsing $\text{Tips}^*(Q)_i$ appear in Q'' 's tally, forcing the carry; any block incompatible with $\text{Tips}^*(Q)_i$ musters at most $2f$ votes, so every carry candidate extends it. The condition $k_i^* = m_i$ cannot be dropped, since without it an equivocating producer can arrange one execution finalising an extension block on one branch, another finalising a proposed entry on the other, with a single V-QC common to both, so that no extraction rule applied to that V-QC is safe in both executions. Below unanimity this fails for $n = 5f + 1$: for any finalisation threshold less than $n - f$, one can construct two executions sharing a V-QC bit-for-bit in which incompatible extensions reach the threshold, so that no extraction rule applied to V-QCs is safe in both (Section 5, where Theorem 4 determines the exact threshold, $\lceil (n + 3f)/2 \rceil$, for every $n \geq 5f + 1$). (Guarantees short of a place in the total ordering are available at weaker thresholds: see the block inclusion theorem of Section 5.)

The total ordering. Suppose T maps each chain to a transaction block on that chain. The *positions above T* are the pairs (o, i) , for offsets $o \geq 1$ and chains $1 \leq i \leq n$; position (o, i) refers to height $T_i.\text{height} + o$ on chain i . The *horizontal order* on these positions is lexicographic with offset taken first: (o, i) precedes (o', i') if $o < o'$, or if $o = o'$ and $i < i'$. (So the positions at offset 1 come first, ordered by chain index, then those at offset 2, and so on.) If T' also maps chains to blocks, then $\text{Horiz}(T, T')$ denotes the sequence of blocks obtained by visiting the positions above T in horizontal order and appending, at each position whose height is at most $T'_i.\text{height}$ on its chain i , the ancestor of T'_i at that height. (In the intended case T'_i extends T_i ; the definition does not require this, however, and when a proposal's anchor jumps to a DA-certificate on a branch incompatible with T_i — possible only under an equivocating producer — the appended blocks are simply read off T'_i 's own path.) Thus every chain's first new block is appended before any chain's second. Note that, for each i , *every* block on T'_i 's path above height $T_i.\text{height}$ appears in the sequence: the definition determines only how the chains are interleaved, never which blocks are included.²² Each V-QC Q then specifies a sequence $\text{Ord}(Q)$ of transaction blocks, defined recursively: $\text{Ord}(Q_{\text{gen}}) := \lambda$ and, where $Q = (W, C)$ is a V-QC for the leader block $\ell = (v, H(Q'), C)$,

$$\text{Ord}(Q) := \text{Ord}(Q') \frown \text{Horiz}(\text{Tips}(Q'), \text{Tips}(Q)).$$

Similarly, each L-QC Q for $\ell = (v, H(Q'), C)$ specifies a sequence

$$\text{Emit}(Q) := \text{Ord}(Q') \frown \rho,$$

defined as follows. Say chain j is *settled* (w.r.t. Q) if $k_j^* = m_j$ and no vote in Q endorses any block strictly extending $\text{Tips}^*(Q)_j$; otherwise j is *unsettled*.²³ To form ρ , visit the positions above $\text{Tips}(Q')$ in horizontal order and, at each position, on chain j say, act as follows:

- if the position's height is at most that of $\text{Tips}^*(Q)_j$, append the ancestor of $\text{Tips}^*(Q)_j$ at that height;
- otherwise, if chain j is settled, append nothing and continue;
- otherwise, halt.

Then ρ is the sequence of blocks appended before the halt. If every chain is settled, no halt ever occurs and ρ is the whole of $\text{Horiz}(\text{Tips}(Q'), \text{Tips}^*(Q))$. In the good case, the entire total ordering, extensions included, is emitted upon receipt of a single L-QC ('entire' meaning through the deepest blocks the voters had received when they voted). The sequence of transactions corresponding to $\text{Emit}(Q)$ (concatenating the values $b.\text{Tr}$ in order, removing duplicates) is what a processor appends to its log upon receiving the L-QC Q . The analysis establishes that $\text{Emit}(Q) \preceq$

²²This is the *horizontal* ordering; heights are measured relative to each chain's previous tip. Alternative interleavings are possible. Section 6 describes one and checks that it preserves the analysis.

²³If j is settled, then no V-QC for the view can specify a tip for chain j beyond $\text{Tips}^*(Q)_j$. Positions cannot advance, being capped at the proposed tip, an ancestor of $\text{Tips}^*(Q)_j$ (or equal to it). An extension carry beyond $\text{Tips}^*(Q)_j$ would require $2f + 1$ votes endorsing a block beyond, hence at least $f + 1$ correct ones, while no vote in Q endorses such a block and at most f correct votes lie outside Q . The condition $k_j^* = m_j$ cannot be dropped, since below the proposed tip a V-QC's tip advances at the threshold $f + 1$, which f faulty votes and a single correct vote from outside Q can meet, and no L-QC can certify that the latter does not exist. An unsettled chain may gain blocks above $\text{Tips}^*(Q)_j$ in the eventual ordering, and blocks so gained would precede, in the horizontal sweep, blocks of other chains at greater offsets. This is why emission must halt at an unsettled chain's frontier. Note that a chain's potential growth is in any case bounded: no vote endorses blocks more than e beyond its reported position, so the sweep's lookahead for an unsettled chain is capped at e beyond the deepest endorsed block: chains are never 'potentially infinite'.

$\text{Ord}(Q'')$ for any V-QC Q'' of the same view, so that logs remain consistent as later views extend the ordering.

Finalisation from the vote pool. Since any $n - f$ votes for ℓ form an L-QC, a processor holding a pool P of at least $n - f$ votes for ℓ , by distinct processors (a faulty processor may sign more than one vote for ℓ ; the pool retains the first it receives, and never replaces it), need never enumerate L-QCs. The rules below apply to P directly, and it is these that the analysis of Section 5 treats, with an L-QC's rules as the special case $|P| = n - f$; they dominate the rules for every individual L-QC assemblable from P , and settledness from a pool is strictly more permissive than from any single L-QC, so pooling can emit strictly more. Formally, we extend the domains of Tips^* , settledness and Emit from L-QCs to pools, the rules below reducing to the rules already given in the case $|P| = n - f$. Define $\text{Emit}(P) := \text{Ord}(Q') \cap \rho$ exactly as for L-QCs, with $\text{Tips}^*(P)$ and settledness with respect to P given, for each chain i , as follows, writing $b_i^* := \text{Tips}^*(P)_i$ for brevity:

- *Positions.* k_i^* is the $(3f + 1)$ -th largest value $\pi(i)$ over the votes in P (this is the drop- $3f$ rule applied to the best $n - f$ votes for chain i);
- *Extensions.* If $k_i^* = m_i$, then b_i^* is the deepest block extending $\text{Prop}(C, i, m_i)$ that at least $n - f$ votes in P endorse (or $\text{Prop}(C, i, m_i)$ itself if there is none). If $k_i^* < m_i$, then $b_i^* := \text{Prop}(C, i, k_i^*)$;
- *Settledness.* Chain i is settled if $k_i^* = m_i$ and $\beta_i + (n - |P|) \leq f$, where β_i is the number of votes in P endorsing some block strictly extending b_i^* .

The settledness condition generalises the L-QC rule (to which it reduces when $|P| = n - f$). It ensures that at most f correct votes endorsing blocks beyond b_i^* can exist at all, seen or unseen, so no V-QC can carry chain i past b_i^* . Emission is monotone, since as votes arrive each quantity above only improves, and successive values of $\text{Emit}(P)$ (like the values $\text{Emit}(Q)$ for the various L-QCs assemblable from P , all of which $\text{Emit}(P)$ dominates) are prefixes of one common sequence, so a processor may re-emit as its pool grows.

The function $\text{SelectAnchor}(\mathcal{S}, v)$. Used by the leader of view v to select the V-QC to build on. If $v' < v$ is the greatest view such that $\mathcal{S}$ contains a V-QC for view v' , the function outputs the V-QC for view v' that p_i forwarded upon its becoming new (the choice among certificates simultaneously assemblable at that timeslot being fixed by the forwarding rule; see ‘New nullifications and V-QCs’ below). The leader thus anchors on an object it has already sent to all.

The procedure $\text{ProposeChains}(Q, v)$. Executed by the leader p_i of view v to determine a new leader block. To execute the procedure, p_i forms the chain proposal C as follows, for each chain j : let $b := \text{Tips}(Q)_j$;

- if $\mathcal{S}$ contains a DA-certificate for a block on chain j of height greater than $b.\text{height}$, then a_j is such a certificate of greatest height; otherwise $a_j := (H(b), b.\text{height})$;
- x_j is the longest sequence of payloads $(H(b^1.\text{Tr}), \dots, H(b^m.\text{Tr}))$, $m \leq d$, such that $b^1, \dots, b^m$ are blocks on chain j that p_i has DA-voted for, with b^1 's parent the base and b^k the parent of b^{k+1} for $k < m$.

Then p_i disseminates the leader block $(v, H(Q), C)$. The V-QC Q requires no dissemination of its own, since **SelectAnchor** returns exactly the object that p_i forwarded to all upon its becoming new.²⁴

When $\mathcal{S}$ contains a valid proposal for view v . This condition is satisfied when:

- (i) $\mathcal{S}$ contains precisely one leader block ℓ of the form $\ell = (v, q, C)$ signed by $\text{lead}(v)$;
- (ii) $\mathcal{S}$ contains a V-QC Q with $H(Q) = q$, for some view $v' < v$, and C is well-formed with respect to $\text{Tips}(Q)$;²⁵ and;
- (iii) $\mathcal{S}$ contains a nullification for each view in the open interval (v', v) .

When (i)–(iii) are satisfied w.r.t. ℓ , we say $\mathcal{S}$ contains a valid proposal ℓ for view v . Note that this condition guards only a processor’s *first* route to voting (line 8 of Algorithm 2). A processor may also vote upon receiving a V-QC for the current view (lines 12–13), *without* checking validity. In this case, the V-QC’s $2f + 1$ votes for the designated proposal ensure that at least $f + 1$ correct processors found it valid, and the V-QC contains everything needed to form the vote (it contains ℓ , which determines $H(\ell)$, and **Positions** and **Extensions** require only C and the voter’s own DA-voting history).

Further local variables. The variable v (initially 1) specifies the present view. The local timer T (initially 0) increments in real time and is reset upon entering each view. The variables **nullified**, **proposed** (initially false) and **notarised** (initially $\perp$) record whether p_i has sent a $\text{nullify}(v)$ message, whether it has proposed a leader block for view v , and the leader block it has voted for in the present view; all are reset upon entering a new view.

New nullifications and V-QCs. As in Minimmit, processors forward all newly received or assembled nullifications and V-QCs to all. At timeslot t , p_i regards a nullification for view v as *new* if $\mathcal{S}$ did not contain one for view v , nor shares forming one, at any smaller timeslot (since threshold signatures are unique, no tie-breaking rule is needed), and regards a V-QC for view v as *new* if $\mathcal{S}$ contained no V-QC for view v at any smaller timeslot, forwarding one such V-QC, chosen by a fixed efficient rule, say assembled greedily from the earliest-received qualifying messages.²⁶

²⁴That the anchor be an object the leader has already sent to all is necessary, not stylistic. A V-QC assembled at proposal time from the leader’s received votes may contain a privately delivered vote, so that no other correct processor holds the exact object; since proposal validity requires the exact object referenced, no correct processor could then vote. Anchoring on the forwarded V-QC rules this out by construction, and since a message sent at timeslot t arrives by $\max\{\text{GST}, t\} + \delta$, the forwarding also delivers the anchor in time for the correct-leader lemma of Section 5. Section 6 relaxes the selection, allowing a better anchor at the price of disseminating it.

²⁵Well-formedness requires, for each chain j : either $a_j = (H(\text{Tips}(Q)_j), \text{Tips}(Q)_j.\text{height})$, or a_j is a valid DA-certificate for chain j of height greater than that of $\text{Tips}(Q)_j$; and $|x_j| \leq d$. Note that $\text{Tips}(Q)$ is computable from Q alone, so this condition is checkable given ℓ and the referenced V-QC. Parent-links among proposed blocks hold by construction (entries are payloads); voters do not verify that the specified blocks *exist*, and simply report a lower position when they have not DA-voted them.

²⁶Unlike Minimmit’s M-notarisations, V-QCs are not constant-size (Section 3.7), but forwarding them is unalarming. In terms of egress, note that under standard operation the leader of the next view in any case sends both a V-QC (its forwarding duty, which doubles as the dissemination of its anchor, per **SelectAnchor**) and its own leader block. Each other processor’s forwarding duty, a single V-QC to all, is therefore less than the per-view egress the protocol already demands of each leader. Moreover, in the common case a processor forwards at a moment when it has nothing else to send, and nothing in the protocol waits on the transmission, so the cost is

Algorithm 1: the chain layer, instructions for p_i

- 1: At every timeslot t :
 - 2: If $\text{Ready}(\mathcal{S})$: ▷ As defined in Section 4
 - 3: $\text{ProduceNext}()$; ▷ Extend own chain
 - 4: For each block b such that p_i may DA-vote for b : ▷ As defined in Section 4
 - 5: Send a DA-vote for b to $p_{b.\text{chain}}$;
 - 6: If $\mathcal{S}$ contains $n - 2f$ DA-votes, with distinct signers, for a block b produced by p_i ,
 - 7: and p_i has not yet disseminated a DA-certificate for b :
 - 8: Form and disseminate the DA-certificate for b ;
-

Algorithm 2: the consensus layer, instructions for p_i

- 1: At every timeslot t :
 - 2: Disseminate new nullifications and new V-QCs in $\mathcal{S}$; ▷ ‘new’ as defined in Section 4
 - 3: If $p_i = \text{lead}(\mathbf{v})$ and $\text{proposed} = \text{false}$:
 - 4: $\text{ProposeChains}(\text{SelectAnchor}(\mathcal{S}, \mathbf{v}), \mathbf{v})$; ▷ Send out a new leader block
 - 5: Set $\text{proposed} := \text{true}$;
 - 6: If $\mathcal{S}$ contains a valid proposal ℓ for view $\mathbf{v}$: ▷ As defined in Section 4
 - 7: If $\text{notarised} = \perp$ and $\text{nullified} = \text{false}$:
 - 8: Set $\text{notarised} := \ell$ and disseminate the vote for ℓ with
 $\pi = \text{Positions}(\mathcal{S}, \ell)$ and $\varepsilon = \text{Extensions}(\mathcal{S}, \ell)$; ▷ Send vote
 - 9: If $T = 2\Delta$, $\text{nullified} = \text{false}$ and $\text{notarised} = \perp$:
 - 10: Set $\text{nullified} := \text{true}$ and disseminate $(\text{novote}, \mathbf{v})$ and $(\text{nullify}, \mathbf{v})$; ▷ Timeout
 - 11: If $\mathcal{S}$ contains a nullification or a V-QC for view $\mathbf{v}$:
 - 12: If $\mathcal{S}$ contains a V-QC for $\mathbf{v}$, designating the proposal of a leader block ℓ ,
 - 13: with $\text{notarised} = \perp$ and $\text{nullified} = \text{false}$: disseminate a vote for ℓ as in line 8;
 - 14: Set $\mathbf{v} := \mathbf{v} + 1$, $\text{nullified} := \text{false}$, $\text{proposed} := \text{false}$,
 $\text{notarised} := \perp$, $T := 0$;
 - 15: ▷ Go to next view
 - 16: If $\text{nullified} = \text{false}$, $\text{notarised} \neq \perp$ and $\mathcal{S}$ contains $\geq 2f + 1$ messages, each signed
 - 17: by a different processor, and each either (i) of the form $(\text{nullify}, \mathbf{v})$ or $(\text{novote}, \mathbf{v})$, or
 - 18: (ii) a vote for some $\ell' \neq \text{notarised}$ for view $\mathbf{v}$:
 - 19: Set $\text{nullified} := \text{true}$ and disseminate $(\text{nullify}, \mathbf{v})$;
 - 20: ▷ Minimmit’s mechanism ensuring view progression; see the walk-through
 - 21: If, for some leader block ℓ , the pool $P \subseteq \mathcal{S}$ of votes for ℓ ,
 by distinct processors, has $|P| \geq n - f$:
 - 22: Finalise ℓ and set $\log_i$ to extend the transactions of $\text{Emit}(P)$;
 - 23: ▷ Pool finalisation as defined in Section 4; monotone, so re-run as P grows
-

Message type	Description
b_{gen}^i	The tuple $(i, 0, \lambda, \lambda)$
transaction block $b \neq b_{\text{gen}}^i$	A tuple (i, h, x, Tr) , signed by p_i on $\text{hd}(b) = (i, h, x, H(\text{Tr}))$
DA-vote for b	$(\text{da}, \text{hd}(b), \rho)$: ρ an $(n - 2f)$ -threshold share on $(\text{da}, \text{hd}(b))$
DA-certificate for b	$(\text{hd}(b), \tau)$: τ the $(n - 2f)$ -threshold signature on $(\text{da}, \text{hd}(b))$
ℓ_{gen}	The tuple $(0, \lambda, \lambda)$
leader block $\ell \neq \ell_{\text{gen}}$	A tuple $(v, H(Q), C)$, signed by $\text{lead}(v)$; $H(\ell) = H(v, H(Q), H(C))$
vote for ℓ	A message $(\text{vote}, v, H(\ell), \pi, \varepsilon, s)$
tally T for ℓ	(E^*, P, D) : describes one vote for ℓ per processor in P
$\text{novote}(v)$	(novote, v, s) : s an ordinary signature on (novote, v)
$\text{nullify}(v)$	$(\text{nullify}, v, \rho)$: ρ a $(2f + 1)$ -threshold share on $(\text{nullify}, v)$
nullification for v	$(\text{nullification}, v, \tau)$: τ the $(2f + 1)$ -threshold signature on $(\text{nullify}, v)$
V-QC for v	(ℓ, T, N, X, s) : certifies $n - f$ votes/novotes, $\geq 2f + 1$ for C
L-QC for ℓ	(ℓ, T, s) : certifies $n - f$ votes for ℓ , distinct signers

Table 1: Messages.

Variable	Description
$\mathbf{v}$	Initially 1, specifies the present view
$\mathbf{T}$	Initially 0, a local timer reset upon entering each view
nullified	Initially false, whether already sent $\text{nullify}(\mathbf{v})$
proposed	Initially false, whether already proposed a leader block for view $\mathbf{v}$
notarised	Initially $\perp$, records leader block voted for in present view
$\mathbf{S}$	Records all received messages, automatically updated Initially contains genesis blocks, certificates and quorums as specified above

Table 2: Local variables. The parameters d (pipelining depth), e (extension bound), B (batch bound) and θ (production interval) are global constants.

Walk-through: lines 16–19 of Algorithm 2 ensure view progression. Section 3 asked the reader to take on trust that every view produces a V-notarisation or a nullification; lines 16–19, inherited from Minimmit, are the delicate part of the mechanism, and we now explain them. The concern is a correct processor that has voted in view $\mathbf{v}$ but sees neither a V-QC nor a nullification form. Perhaps the leader equivocated and votes are split, or other processors timed out before the proposal reached them. Waiting indefinitely is not an option, but nor may the processor simply nullify, since a nullification must prove that *no* view $\mathbf{v}$ block receives an L-notarisation, and the processor’s own vote may be helping some block towards exactly that. The rule resolves the tension: the processor nullifies only upon receiving, from $2f + 1$ distinct processors, messages none of which support the block ℓ it voted for (nullify or novote messages, or votes for other blocks). At least $f + 1$ of these come from correct processors and, considering the *first* correct supporter of any given block to nullify in this way, one can check that none of those correct processors ever votes for ℓ , capping its support at $n - f - 1$, so nullifying is safe. Conversely, the rule ensures progression. After GST, every correct processor eventually holds a view $\mathbf{v}$ message from every correct processor. If $2f + 1$ votes for a single block are among them, a V-notarisation exists, and otherwise every correct voter eventually accumulates $2f + 1$ messages

bandwidth off the critical path. In terms of ingress, a processor already holding the V-QC need not attend to more than a brief identifier at the head of the message.

Notion	Description
$\text{Prop}(C, i, k)$	The block specified at position k on chain i by chain proposal C (position 0 being the base)
m_i	The number of payload entries C makes for chain i ; the proposed tip is $\text{Prop}(C, i, m_i)$
endorsed tip	Of a vote, per chain: the last block its extensions specify, or its position block $\text{Prop}(C, i, \pi(i))$ if none
endorses	A vote endorses b if b is an ancestor of, or equal to, its endorsed tip on b 's chain
$\text{Tips}(Q)$	Safe-to-extend tips of the V-QC Q : per chain, the $(f+1)$ -th largest reported position, then the deepest block extending it that $2f+1$ tally votes endorse
k_i^*	The $(3f+1)$ -th largest reported position for chain i , over an L-QC's votes (or a pool)
$\text{Tips}^*(Q), \text{Tips}^*(P)$	Finalised tips of the L-QC Q , or of the pool P (abbreviated $b_i^* := \text{Tips}^*(P)_i$): per chain, $\text{Prop}(C, i, k_i^*)$ and, if $k_i^* = m_i$, the deepest block extending it that <i>every</i> vote endorses ($n - f$ votes, for a pool)
settled	Chain i , w.r.t. an L-QC or pool: $k_i^* = m_i$ and at most f correct votes endorsing blocks beyond b_i^* can exist ($\beta_i + (n - P) \leq f$)
$\text{Horiz}(T, T')$	The blocks above T on T' 's paths, visited by offset and then by chain index
$\text{Ord}(Q)$	The total ordering extracted from the V-QC Q : recursively, Ord of the referenced V-QC followed by Horiz towards $\text{Tips}(Q)$
$\text{Emit}(Q), \text{Emit}(P)$	The emitted prefix: Ord of the referenced V-QC, then the sweep towards the finalised tips, halting at the first empty slot of an unsettled chain

Table 3: The extraction apparatus, as defined in this section.

of the stated forms and nullifies, so that a nullification forms. Note finally that the processor sends its nullify share *without* a novote message, since it did vote and a novote would be false. (Full arguments appear in Section 5.)

5 Verification

5.1 Consistency

Following Minimmit, we reason about *notarisations*: sets of messages actually sent, whether or not any processor assembles the corresponding certificate. Precisely, we say a transaction block b is *certifiable* if at least $n - 2f$ processors send DA-votes for b ; a leader block ℓ *receives an L-notarisation* if at least $n - f$ processors send votes for ℓ ; a *V-notarisation* for view v is a set W of view v votes and $\text{novote}(v)$ messages, one each from $n - f$ distinct processors, at least $2f + 1$ of which are votes for a common leader block ℓ , whose proposal W is said to *designate*; and view v *receives a nullification* if at least $2f + 1$ processors send $\text{nullify}(v)$ messages.²⁷ Any

²⁷One caution regarding designation: for $n = 5f + 1$ a V-notarisation designates exactly one proposal (two disjoint sets of $2f + 1$ votes would require $4f + 2 > n - f$ messages), but for larger n it may designate two. The ambiguity is harmless. A V-QC always names a *single* designated proposal, that of the leader block it contains, and it is this designation that parenthood (below) and the protocol's rules consult. Wherever the lemmas below extract values from a V-notarisation (Lemmas 7 and 8), an L-notarisation for the view is in force, and Lemma 4

QC certifies a notarisation of the corresponding kind (this uses the assumption that signatures cannot be forged), and the functions **Tips**, **Tips***, **Ord** and **Emit** depend only on the underlying notarisation and the designated proposal, so all statements below transfer to QCs. Recall from Section 2 that a message received at timeslot t was sent at a timeslot strictly before t .

We begin with the chain layer.

Lemma 1 (Certified blocks are compatible). *Any two certifiable transaction blocks on the same chain are compatible.*

Proof. Towards a contradiction, suppose otherwise. Among all pairs of incompatible certifiable blocks (on some fixed chain), choose a pair b, b' , of heights $h \leq h'$, so that (h, h') is lexicographically minimal. Each of b, b' has DA-votes from $n - 2f$ distinct processors. The two voter sets intersect in at least $2(n - 2f) - n = n - 4f \geq f + 1$ processors, so some correct processor p must DA-vote for both. If $h = h'$ this contradicts condition (i) for DA-voting (one DA-vote per chain and height), so $h < h'$.

Consider the timeslot at which p sent its DA-vote for b' , and let b'' , of height $h'' < h'$, be the block certified by the anchoring certificate at that vote (so that h'' is the greatest certified height below h' in p 's local value $\mathcal{S}$ at that timeslot). By condition (iii), p then held blocks $b_{h''+1}, \dots, b_{h'} = b'$ forming the path from b'' to b' .

First, suppose $h'' \geq h$. The block b'' is certifiable, and the pair b, b'' has both heights below h' . It follows from our choice of (h, h') that the blocks b and b'' are compatible. Then b is an ancestor of b'' , hence of b' , giving the required contradiction.

So $h'' < h$, and we may set z to be the block on the path from b'' to b' of height h . Since b and b' are incompatible, b and z are incompatible; in particular $z \neq b$. By condition (iii), p sent a DA-vote for z at or before the timeslot at which it voted for b' . Since p also DA-votes for b , this means p DA-votes at height h for two distinct blocks, contradicting condition (i). $\square$

We next turn to the consensus layer.

Lemma 2 (One vote per view). *A correct processor sends at most one vote for each view.*

Proof. Votes are sent only at lines 7–8 and 12–13 of Algorithm 2, both guarded by **notarised** = $\perp$; voting sets **notarised** to the block voted for, and the variable is reset only upon entering a new view. $\square$

Lemma 3 (Vote–novote exclusivity). *No correct processor sends both a vote for a view v leader block and a $\text{novote}(v)$ message.*

Proof. Novote messages are sent only at line 10, guarded (at line 9) by **notarised** = $\perp$ and **nullified** = false. A prior vote in the view sets **notarised** $\neq \perp$, blocking line 10. A novote sets **nullified** = true, blocking both voting guards for the remainder of the view. $\square$

Lemma 4 (Designation). *Suppose the view v leader block ℓ receives an L -notarisation. Then at most $2f$ processors ever send view v votes for leader blocks other than ℓ , or send $\text{novote}(v)$ messages. In particular, every view v V -notarisation designates ℓ 's proposal, and no other view v leader block receives $2f + 1$ votes.*

then makes the designation unique.

Proof. At least $n - 2f$ of the $n - f$ processors voting for ℓ are correct and, by Lemmas 2 and 3, send no other view v vote and no $\text{novote}(v)$ message. Any view v vote for another block, or any $\text{novote}(v)$ message, is therefore sent by one of the remaining at most $2f$ processors. $\square$

Lemma 5 (No nullification). *Suppose the view v leader block ℓ receives an L-notarisation. Then view v receives no nullification.*

Proof. Let A be the set of correct processors that vote for ℓ and let B be the set of remaining correct processors. If $f_a \leq f$ is the number of faulty processors, $|A| \geq n - f - f_a$, so $|B| \leq (n - f_a) - (n - f - f_a) = f$.

We claim no member of A ever sends $\text{nullify}(v)$. A member of A has **notarised** $\neq \perp$ for the remainder of the view, so can send $\text{nullify}(v)$ only via lines 16–19, upon holding $2f + 1$ messages by distinct processors, each a $\text{nullify}(v)$, a $\text{novote}(v)$, or a view v vote for a block other than ℓ . Towards a contradiction, suppose there is some least timeslot t at which some $p \in A$ does so. Each of p 's $2f + 1$ evidence messages was sent strictly before t . Consider any correct sender among them: a $\text{novote}(v)$ sender lies in B by Lemma 3; the sender of a vote for a block other than ℓ lies in B by Lemma 2; and a $\text{nullify}(v)$ sender lies in B , or else is a member of A that sent $\text{nullify}(v)$ strictly before t , contradicting the minimality of t . So the correct senders all lie in B , and number at most f . With at most f faulty senders, the evidence totals at most $2f < 2f + 1$: a contradiction.

Correct $\text{nullify}(v)$ senders therefore all lie in B . With at most f faulty processors, fewer than $2f + 1$ processors send $\text{nullify}(v)$ messages, and view v receives no nullification. $\square$

Next, we establish consistency of the leader chain itself. Define the *parent* of a leader block $\ell = (v, H(Q), C)$ to be the leader block that Q contains (and define the parent of any leader block referencing Q_{gen} to be ℓ_{gen}). Ancestry and compatibility for leader blocks are then defined as for transaction blocks.

Lemma 6 (The leader chain). *Suppose the view v leader block ℓ receives an L-notarisation. Then every leader block receiving $2f + 1$ votes in any view $v' \geq v$ has ℓ as an ancestor. In particular, leader blocks receiving L-notarisations are pairwise compatible.*

Proof. By induction on v' . For $v' = v$ the claim is Lemma 4. So let $v' > v$, suppose the claim holds for all views in $[v, v')$, and let $\ell' = (v', H(Q''), C')$ receive $2f + 1$ votes, at least $f + 1$ of them from correct processors. Let p_j be a correct processor voting for ℓ' at the earliest timeslot, t_j say. If p_j voted via lines 12–13, its $\mathcal{S}$ contained a V-QC for view v' itself, designating ℓ' 's proposal. (This rule fires only upon a V-QC for the processor's current view; this is not the earlier-view V-QC Q'' that ℓ' references, which belongs to the line 8 route below.) Such a V-QC certifies $2f + 1$ view v' votes for ℓ' , at least $f + 1$ of them correct, all sent strictly before t_j . This contradicts the choice of p_j . So p_j voted via lines 7–8, and its $\mathcal{S}$ contained a valid proposal, in particular the V-QC Q'' , for some view $v'' < v'$, together with a nullification for every view in the open interval (v'', v') . A nullification for a view exists only if that view receives one ($2f + 1$ shares are needed to form the threshold signature, and shares by correct processors cannot be forged), so by Lemma 5 we have $v \notin (v'', v')$, i.e. $v'' \geq v$. Now the tally of Q'' certifies $2f + 1$ view v'' votes for ℓ' 's parent ℓ'' , which therefore receives $2f + 1$ votes in view $v'' \in [v, v')$. By the induction hypothesis (or by Lemma 4 if $v'' = v$), ℓ is an ancestor of ℓ'' , hence of ℓ' .

For the final claim, if ℓ and ℓ' receive L-notarisations in views $v \leq v'$, then ℓ' in particular receives $2f + 1$ votes, so has ℓ as an ancestor. $\square$

We state the next lemma for pools. Taking P to be the votes of an L-QC recovers the statement for Tips^* .

Lemma 7 (Safe extension). *Let the leader block $\ell = (v, H(Q), C)$ receive an L-notarisation, let P be a set of votes for ℓ by at least $n - f$ distinct processors, and let $k_i^*, b_i^* := \text{Tips}^*(P)_i$ and settledness with respect to P be as in ‘Finalisation from the vote pool’ (Section 4). Let W be any view v V-notarisation. Then, for every chain i :*

- (a) $\text{Tips}(W)_i$ extends b_i^* , and;
- (b) if chain i is settled w.r.t. P , then $\text{Tips}(W)_i = b_i^*$.

Proof. By Lemma 4, W designates ℓ ’s proposal, so $\text{Tips}(W)$ is computed from the tally V_i of W ’s votes for ℓ , relative to the same chain proposal C . A correct processor appearing in W appears with its unique actual vote, in V_i if it voted for ℓ (in that case it cannot contribute a novote by Lemma 3, nor a vote for another block by Lemma 2). Fix a chain i , and let $f_a \leq f$ be the number of faulty processors.

Positions. By definition of k_i^* , at least $3f+1$ votes in P , by distinct processors, have $\pi(i) \geq k_i^*$; at least $2f+1$ of these processors are correct, and at most f processors have no message in W , so at least $f+1$ votes in V_i have $\pi(i) \geq k_i^*$. The $(f+1)$ -th largest position in V_i is therefore at least k_i^* . Writing k_i for the position rank of $\text{Tips}(W)_i$, we have $k_i \geq k_i^*$.

Proof of (a). If $b_i^* = \text{Prop}(C, i, k_i^*)$, then the position block $\text{Prop}(C, i, k_i)$ extends b_i^* , and the extension-carry step only ever replaces a block by one extending it, so $\text{Tips}(W)_i$ extends b_i^* . Otherwise $k_i^* = m_i$ and at least $n - f$ votes in P , by distinct processors, endorse b_i^* , which strictly extends $\text{Prop}(C, i, m_i)$. Then $k_i = m_i$ (positions are capped at m_i), so the position block is $\text{Prop}(C, i, m_i)$, an ancestor of b_i^* . At least $n - f - f_a$ of the processors endorsing b_i^* are correct, and at least $n - f - f_a - f \geq n - 3f \geq 2f + 1$ of them appear in V_i , endorsing b_i^* , so b_i^* is a carry candidate. Moreover every carry candidate is compatible with b_i^* : if b is incompatible with b_i^* , then a correct processor endorsing b has DA-voted a block incompatible, at some height, with the one DA-voted by each of the $\geq n - f - f_a$ correct processors endorsing b_i^* , so (correct processors DA-voting once per height) at most $(n - f_a) - (n - f - f_a) = f$ correct processors endorse b . Therefore b gains at most $f + f_a \leq 2f$ votes in V_i , which is below the carry threshold. A deepest candidate is at least as deep as b_i^* and compatible with it, so extends it, whichever candidate the tie-breaking rule selects. $\text{Tips}(W)_i$ therefore extends b_i^* .

Proof of (b). Suppose chain i is settled, meaning $k_i^* = m_i$ and $\beta_i + (n - |P|) \leq f$. Any correct processor whose vote endorses a block strictly extending b_i^* either has its vote in P (there are at most β_i such) or has no vote in P (there are at most $n - |P|$ such processors), giving at most f correct processors in all. A carry candidate b' strictly extending b_i^* would need $2f + 1$ votes in V_i endorsing b' , of which at most f correct and at most $f_a \leq f$ faulty exist, so no such b' exists. Candidates incompatible with b_i^* are excluded as in (a), and positions are capped at m_i , so every candidate, and the position block, is an ancestor of or equal to b_i^* . By (a), $\text{Tips}(W)_i$ extends b_i^* . Combining, $\text{Tips}(W)_i = b_i^*$. $\square$

Lemma 8 (Emission is a prefix). *In the setting of Lemma 7, $\text{Emit}(P) \preceq \text{Ord}(Q')$ for every V-QC Q' for view v . In particular, any two values $\text{Emit}(P)$, $\text{Emit}(P')$ extracted from pools of votes for ℓ (by different processors, or at different times) are compatible.*

Proof. By Lemma 4, the leader block contained in Q' is ℓ , so Q' references Q and $\text{Ord}(Q') = \text{Ord}(Q) \cap \text{Horiz}(\text{Tips}(Q), \text{Tips}(Q'))$, while $\text{Emit}(P) = \text{Ord}(Q) \cap \rho$: it suffices to show that $\rho \preceq$

$\text{Horiz}(\text{Tips}(Q), \text{Tips}(Q'))$. Note that Lemma 7 applies to Q' , taking W to be the V-notarisation that Q' certifies. Both sequences visit the positions above $\text{Tips}(Q)$ in horizontal order. Consider any position visited by ρ strictly before its halt (if any), on chain j say. If the position's height is at most that of b_j^* , then ρ appends the ancestor of b_j^* at that height; by Lemma 7(a) this is also the ancestor of $\text{Tips}(Q')_j$ at that height, which is what $\text{Horiz}(\text{Tips}(Q), \text{Tips}(Q'))$ appends. Otherwise chain j is settled (else ρ would have halted here) and, by Lemma 7(b), $\text{Tips}(Q')_j = b_j^*$ lies below the position, so both sequences append nothing. The two sequences thus agree at every position up to ρ 's halt, where ρ ends: $\rho \preceq \text{Horiz}(\text{Tips}(Q), \text{Tips}(Q'))$.

For the final claim, note that a V-notarisation for view v exists: any $n - f$ votes of the L-notarisation received by ℓ form one. Both $\text{Emit}(P)$ and $\text{Emit}(P')$ are prefixes of $\text{Ord}(Q')$ for a V-QC Q' certifying it. $\square$

Theorem 1 (Consistency). *If p_i and p_j are correct then, for any timeslots t and t' , $\log_i(t)$ and $\log_j(t')$ are compatible.*

Proof. Every value taken by the log of a correct processor is the transaction sequence of $\text{Emit}(P)$ for some pool P of $n - f$ or more votes for a leader block ℓ (lines 21–22 of Algorithm 2). In particular ℓ receives an L-notarisation. Since prefixes of block sequences yield prefixes of the corresponding deduplicated transaction sequences, it suffices to show that any two such values $\text{Emit}(P)$, $\text{Emit}(P')$, for leader blocks ℓ and ℓ' of views $v \leq v'$, are compatible as block sequences.

If $v = v'$ then $\ell' = \ell$ by Lemma 4, and the claim is Lemma 8. So suppose $v < v'$. By Lemma 6, ℓ is an ancestor of ℓ' . Let $\ell = \ell_0, \ell_1, \dots, \ell_r = \ell'$ be the path, and for each $s < r$ let $Q^{(s)}$ be the V-QC referenced by ℓ_{s+1} , so that $Q^{(s)}$ designates the proposal of ℓ_s . Then:

- $\text{Emit}(P) \preceq \text{Ord}(Q^{(0)})$, by Lemma 8, since $Q^{(0)}$ is a V-QC for view v ;
- $\text{Ord}(Q^{(s)}) \preceq \text{Ord}(Q^{(s+1)})$ for each $s < r-1$, since ℓ_{s+1} , the leader block contained in $Q^{(s+1)}$, references $Q^{(s)}$, so that $\text{Ord}(Q^{(s+1)})$ extends $\text{Ord}(Q^{(s)})$ by definition, and;
- $\text{Ord}(Q^{(r-1)}) \preceq \text{Emit}(P')$, since ℓ' references $Q^{(r-1)}$ and $\text{Emit}(P') = \text{Ord}(Q^{(r-1)}) \cap \rho'$.

Chaining, $\text{Emit}(P) \preceq \text{Emit}(P')$. $\square$

5.2 Liveness

Recall from Section 4 that $\mathcal{S}$ contains a certificate (a V-QC, L-QC, nullification or DA-certificate) whenever it contains messages from which one can be formed. We add one reading convention: the instructions of Algorithms 1 and 2 are executed repeatedly within each timeslot, until no instruction applies, so that a processor holding view-advancing triggers for several consecutive views passes through all of them in a single timeslot. Bounds below are stated in terms of δ , the (unknown) least upper bound on message delay after GST (Section 2), wherever possible: a message sent at timeslot t arrives by $\max\{\text{GST}, t\} + \delta$, and the known bound $\Delta \geq \delta$ enters only through the 2Δ timeouts.

Lemma 9 (View synchronisation). *If the first correct processor p to enter view v does so at timeslot t , then every correct processor enters a view $\geq v$ by $\max\{\text{GST}, t\} + \delta$.*

Proof. Processor p passed through every view $v' < v$, so its local value $\mathcal{S}$ at t contains, for each such v' , a nullification or V-QC for v' . For each such v' it has therefore forwarded a nullification or V-QC for v' to all processors at the timeslot $\leq t$ at which one first became new to it (line 2 of Algorithm 2). All of these messages arrive by $\max\{\text{GST}, t\} + \delta$. A correct processor holding,

for every $v' < v$, a nullification or V-QC for v' advances (lines 11 and 14), within the timeslot, to a view $\geq v$. $\square$

Lemma 10 (View exit). *Suppose the first correct processor to enter view v does so at timeslot $t \geq \text{GST}$. Then every correct processor enters a view $\geq v + 1$ by $t + 2\Delta + 3\delta$.*

Proof. Let t' be the first timeslot at which a correct processor enters a view $\geq v + 1$ (if ever). If $t' \leq t + 2\Delta + 2\delta$ then, by Lemma 9, we are done. So suppose no correct processor enters a view $\geq v + 1$ by $t + 2\Delta + 2\delta$. We show $t' \leq t + 2\Delta + 3\delta$.

By Lemma 9, every correct processor enters view v by $t + \delta$, and, remaining in the view, sends exactly one of the following by $t + 2\Delta + \delta$: a vote (lines 7–8 or 12–13), or, at 2Δ on its local timer, a novote(v) and nullify(v) pair (lines 9–10). By $t + 2\Delta + 2\delta$, therefore, every correct processor holds a view v vote or novote from every correct processor, giving at least $n - f$ view v messages by distinct processors. There are two cases.

Suppose first that at least $2f + 1$ correct processors vote for a common leader block ℓ . Then by $t + 2\Delta + 2\delta$ the $\mathcal{S}$ of every correct processor contains a V-QC for view v (assembled from $n - f$ of the correct messages), and every correct processor enters view $v + 1$ at $t + 2\Delta + 2\delta$ (lines 11 and 14), a contradiction to our supposition.

Otherwise, every leader block receives at most $2f$ correct votes. Consider any correct processor p that voted, for ℓ_p say. Of the $\geq n - f$ correct view v messages that p holds by $t + 2\Delta + 2\delta$, those that are votes for ℓ_p number at most $2f$, so at least $n - 3f \geq 2f + 1$, by distinct processors, are each a nullify(v), a novote(v), or a vote for a leader block other than ℓ_p . At $t + 2\Delta + 2\delta$, p then sends nullify(v) via lines 16–19, if it has not already done so. Every correct processor has thus sent nullify(v) by $t + 2\Delta + 2\delta$. The shares arrive by $t + 2\Delta + 3\delta$, every correct processor assembles a nullification, and enters view $v + 1$. $\square$

Since the first correct processor to enter view $v + 1$ does so at or after the first correct entry into view v , an induction gives:

Corollary 1. *Every view is eventually entered by every correct processor.*

Lemma 11 (Correct-leader views). *Suppose $\text{lead}(v)$ is correct and the first correct processor to enter view v does so at timeslot $t \geq \text{GST}$. Then every correct processor votes for the leader's block ℓ by $t + 2\delta$, so that ℓ receives an L -notarisation. By $t + 3\delta$ every correct processor holds a pool of at least $n - f$ votes for ℓ , so finalises ℓ (lines 21–22) and enters view $v + 1$.*

Proof. By Lemma 9, every correct processor enters view v by $t + \delta$. Entry times thus lie in $[t, t + \delta]$, so no correct processor's view v timer reaches 2Δ before $t + 2\Delta$.

Every correct processor holds a valid proposal by $t + 2\delta$. The leader enters view v at some $t' \leq t + \delta$ and immediately proposes (lines 3–5), disseminating $\ell = (v, H(Q), C)$ for the V-QC Q returned by `SelectAnchor`, of view v' say, and ℓ arrives everywhere by $t' + \delta \leq t + 2\delta$. Since `SelectAnchor` returns a V-QC for the *greatest* view for which the leader holds one, the leader advanced through each view $u \in (v', v)$ by nullification, and holds a nullification for each. The V-QC Q was forwarded to all by the leader itself upon its becoming new, at some timeslot $\leq t'$ (`SelectAnchor` returns exactly the forwarded object), and so arrives everywhere by $\max\{\text{GST}, t'\} + \delta \leq t + 2\delta$; each nullification became new to the leader at some timeslot $\leq t'$ and was forwarded to all upon becoming new (line 2, by the leader or by whoever first supplied it), so arrives everywhere by $\max\{\text{GST}, t'\} + \delta \leq t + 2\delta$ (threshold signatures being unique, what was forwarded is the nullification). The leader being correct, ℓ is the unique view v leader block

signed by $\text{lead}(v)$ and C is well-formed with respect to $\text{Tips}(Q)$: conditions (i)–(iii) of proposal validity hold at every correct processor by $t + 2\delta$.

No correct processor sends a $\text{novote}(v)$ or $\text{nullify}(v)$ message. Timeouts (lines 9–10) fire no earlier than $t + 2\Delta \geq t + 2\delta$, by which time $\mathcal{S}$ contains the valid proposal, and lines 6–8 precede lines 9–10 within a timeslot, so a processor reaching 2Δ on its timer votes rather than timing out (its guards hold, as we check below). For $\text{nullify}(v)$ via lines 16–19: consider the first correct processor to send $\text{nullify}(v)$. It has voted, necessarily for ℓ (a vote via lines 7–8 requires a valid proposal, and ℓ is the unique candidate; a vote via lines 12–13 requires a view v V-QC designating another block, hence $f + 1$ correct votes for that block, which do not exist, by this same argument applied inductively to earlier voters). Its evidence must contain $2f + 1$ messages, none supporting ℓ . Correct processors supply no novotes and no earlier nullifies (it is first), and no votes for other blocks, so at most f faulty messages qualify, i.e., the evidence never accumulates.

Every correct processor votes for ℓ by $t + 2\delta$. At the first timeslot ($\leq t + 2\delta$) at which it holds the valid proposal while in view v , its guards hold: $\text{nullified} = \text{false}$ since it sends no $\text{nullify}(v)$ (above); $\text{notarised} = \perp$ unless it has already voted for ℓ (the unique votable block, as above). It remains to check that it is still in view v . It cannot have entered $v + 1$ via a nullification for v ($f + 1$ correct shares would be needed, and none exist), and if via a V-QC for v , then lines 12–13 fire before line 14, so it voted for ℓ first.

Thus at least $n - f$ correct processors vote for ℓ by $t + 2\delta$, providing an L-notarisation. The votes arrive at all correct processors by $t + 3\delta$, whereupon every correct processor finalises ℓ (lines 21–22) and, holding a V-QC for view v (any $n - f$ of the votes), enters view $v + 1$. $\square$

Lemma 11 is the 3δ figure of Section 3.6. In a correct-leader view after GST, the 2Δ timeouts never bind, and every bound is governed by the actual network delay.

Data availability. One convention remains before the theorem. Recall from Section 2 that we solve Extractable SMR: the protocol guarantees that the data of every block entering the ordering is retrievable, with the retrieval mechanism left to implementation. Note first that $\text{Emit}(P)$ is computable without block data: votes, chain proposals and headers determine the emitted sequence of block *identifiers*. We accordingly read lines 21–22 modulo retrieval: the emitted identifiers are determined at once, and the corresponding transactions enter the log as the data of each emitted block is obtained. Retrieval terminates, because every block of an emitted sequence, together with its ancestors, is held in full by at least one correct processor, and by at least $2f + 1$ whenever the block lies at or below a finalised tip. (Briefly: at least $2f + 1$ correct processors DA-voted at or above any finalised tip; a DA-vote requires possession of the block and of its uncertified ancestors down to a certified block; and every certified block, with its ancestors, is held by at least $2f + 1$ correct DA-voters, by an induction descending the chain. The weaker one-holder bound concerns only blocks carried by safe-to-extend tips, whose $f + 1$ endorsing votes guarantee a single correct holder, on branches later abandoned: possible only under an equivocating producer. The availability lemma of the next subsection gives the full accounting.) A correct processor missing the data of an emitted block requests it from all processors, and after GST a correct holder answers within 2δ . The same convention covers the V-QCs of the reference chain, which computing Ord requires and which a processor finalising via the pool need not hold: a processor missing a V-QC named by hash requests it from all. Possession is guaranteed, since any leader block on the reference chain of a finalised block received $2f + 1$ votes, and the first $f + 1$ correct processors to vote for it necessarily did so via the valid-proposal route (a vote via lines 12–13 presupposes a view V-QC containing $f + 1$

earlier correct votes), each therefore holding the referenced V-QC. In the common case no such retrieval occurs, each view's referenced V-QC having been forwarded to all upon first receipt, by the leader among others.

Theorem 2 (Liveness). *If p_i and p_j are correct and p_i receives the transaction tr then, for some t , $\text{tr} \in \log_j(t)$.*

Proof. We may suppose tr never enters $\log_j$, and derive a contradiction. Throughout, we use the fact that all delays after GST are bounded, so that ‘eventually’ claims compose.

Step 1: tr enters p_i 's chain. We first observe that every block p_i produces is eventually DA-certified, with the certificate eventually held by every correct processor. By induction on height h : once every correct processor holds the certificate for p_i 's block at height $h - 1$ together with p_i 's block b_h (which p_i disseminated), the block b_h is eligible at every correct processor: condition (i) holds since a correct processor's DA-votes on chain i are all for blocks of p_i 's unique chain (p_i signs one block per height, and forgeries are excluded), so no conflicting vote exists; (ii) holds with $h - h' \leq 1$; and (iii) holds along p_i 's chain, the votes below b_h having been sent in previous steps of the induction. Correct processors DA-vote every eligible block (line 4 of Algorithm 1), so p_i receives $n - 2f$ shares, forms the certificate, and disseminates it. Consequently the window condition of **Ready** is satisfied within bounded time whenever it fails, and, since $\text{tr} \in \text{pending}$ persistently (it is never included, else Steps 2–3 below apply to the including block), the pacing condition ensures that p_i eventually executes **ProduceNext**, producing a block b containing tr , at height h_b say.

Step 2: b is finalised. Let $h^* \geq h_b$ be the height of p_i 's tip at some fixed timeslot. Choose $t^* \geq \text{GST}$ large enough that, by t^* , every correct processor has received and DA-voted every block of chain i up to height h^* , and holds every DA-certificate for chain i up to height $h^* - d$. (Such certificates exist: by the window condition, p_i produced its block at height h^* only with a certified block at height at least $h^* - d$, and certificates are disseminated.) By Corollary 1 and the round-robin **lead** function, some view v with correct leader has its first correct entry at a timeslot $t \geq t^*$. By Lemma 11, every correct processor votes for the leader's block ℓ by $t + 2\delta$ and finalises it by $t + 3\delta$.

Now consider the proposal's chain i coordinate. The leader anchors at $\text{Tips}(Q)_i$ or at a DA-certificate of greater height, whichever is higher. It holds certificates for chain i through height $h^* - d$, so the base has height at least $h^* - d$. The entries comprise every block the leader has DA-voted extending the base, up to d of them, and the leader has DA-voted chain i through height h^* . The proposed tip therefore has height at least $\min\{(h^* - d) + d, h^*\} = h^* \geq h_b$. If the base itself has height at least h^* , then the finalised tip extends the base, which extends b on the chain's single path, and we are done. Otherwise every proposed entry at height at most h^* lies on the single path and was DA-voted by every correct processor by t^* , so every correct voter reports a position of height at least h^* . (The leader may also propose entries above h^* , which correct voters need not hold; this only lowers reported positions towards the height- h^* entry, never below it.) The $(3f + 1)$ -th largest position over the $n - f$ votes of any L-notarisation for ℓ therefore has height at least $h^* \geq h_b$, and the finalised tip extends the block at that position. Since the blocks of chain i form a single path, b is an ancestor of the view v finalised tip on chain i .

Step 3: b is emitted. The value $\text{Emit}(P)$ computed at view v may halt before reaching b 's slot, since some other chain may be unsettled. Let $v' > v$ be the next view with correct leader, with first correct entry at a timeslot $t' \geq t$ (Corollary 1 again). By Lemma 11, every correct processor finalises the view v' block $\ell_{v'}$ by $t' + 3\delta$. In doing so it computes $\text{Emit}(P') = \text{Ord}(Q^*) \cap \rho'$, where

Q^* is the V-QC referenced by $\ell_{v'}$. Now ℓ receives an L-notarisation, so ℓ is an ancestor of $\ell_{v'}$ by Lemma 6. The chain of references from $\ell_{v'}$ therefore passes through a V-QC Q_v for view v , and $\text{Ord}(Q^*) \supseteq \text{Ord}(Q_v)$. By Lemma 7, $\text{Tips}(Q_v)_i$ extends the view v finalised tip on chain i , hence extends b . Each segment $\text{Horiz}(T, T')$ of the recursion defining Ord includes every block on T' 's path in the heights it covers, so b appears in $\text{Ord}(Q_v)$, hence in $\text{Ord}(Q^*)$, hence in $\text{Emit}(P')$. Every correct processor therefore emits a sequence of identifiers containing b by $t' + 3\delta$, plus bounded time for any reference-chain V-QCs it must retrieve, and by the retrieval convention it appends the corresponding transactions within bounded further time. (For b itself no retrieval is needed, since every correct processor holds b , having DA-voted it.) Either tr was already in its log, as a removed duplicate, or it enters now. In either case $\text{tr} \in \log_j$, a contradiction. $\square$

5.3 Remaining properties

We collect the remaining claims made in earlier sections: the availability lemma promised in Section 5.2, membership finality, the block inclusion theorem, and the lower bound forcing the extraction thresholds. Throughout, ‘held in full’ means possession of a block’s complete data (i, h, x, Tr) , as opposed to its header or hash, and $f_a \leq f$ denotes the number of actually faulty processors in the execution.

Availability.

- Lemma 12** (Availability). *(a) Every certifiable transaction block, and each of its ancestors, is held in full by at least $2f + 1$ correct processors (the set of holders may vary from block to block).*
- (b) For every V-notarisation W and chain i , the block $\text{Tips}(W)_i$, and each of its ancestors, is held in full by at least one correct processor.*
- (c) Every block of an emitted sequence $\text{Emit}(P)$ is held in full by at least one correct processor; and by at least $2f + 1$ if, on its chain, it lies at or below the finalised tip of some emission, where that tip certifies something new: a position block of rank $k^* \geq 1$, a DA-certified base, or a unanimously endorsed extension block. (The excluded case is $k^* = 0$ with a hash-anchor base: the finalised tip is then the anchor itself, merely re-affirming the previous safe-to-extend tip, whose endorsing votes DA-voted nothing; blocks covered by no other finalisation retain only the single-holder guarantee, which suffices for retrieval.)*

Proof. (a) Let b be an ancestor of a certifiable block and, among the certifiable blocks extending b (or equal to it), let b^* be one of least height. At least $n - 2f$ processors DA-voted b^* , so at least $n - 3f \geq 2f + 1$ correct ones. By condition (iii), each possessed, when it voted, the path from its anchor (a certifiable ancestor of b^* of smaller height) to b^* . No such anchor has height at least that of b , since it would then be a certifiable block extending b (anchors and b are ancestors of b^* , hence comparable) of height smaller than b^* 's, contradicting minimality. So each of the $2f + 1$ correct voters held b .

(b) We argue by induction on views. If $\text{Tips}(W)_i$ is a position block $\text{Prop}(C, i, k_i)$ with $k_i \geq 1$, then at least $f + 1$ votes in the tally report $\pi(i) \geq k_i$, so at least one correct processor DA-voted positions $1, \dots, k_i$, and held, at each vote, the corresponding block and the path down to its certifiable anchor. If the extension carry applies, at least $2f + 1$ votes endorse $\text{Tips}(W)_i$, so at least $f + 1$ correct processors did the same along the carried path. In either case, ancestors below the relevant anchors are covered by (a). If $k_i = 0$ and no carry applies, then $\text{Tips}(W)_i$ is the

base itself, which well-formedness requires to be either a DA-certificate, covered by (a), or a hash anchor equal to $\text{Tips}(Q')_i$ for the V-QC Q' referenced by the proposal, which is for an earlier view, and the induction hypothesis applies (the base case being genesis, held by all).

(c) For the stronger claim, suppose the block lies at or below a finalised tip b_j^* of one of the three kinds hypothesised, and argue relative to that emission's pool. If $b_j^* = \text{Prop}(C, j, k_j^*)$ with $k_j^* \geq 1$, then at least $3f + 1$ votes report $\pi(j) \geq k_j^*$, so at least $2f + 1$ correct processors DA-voted positions $1, \dots, k_j^*$; each held any given ancestor b of b_j^* directly, or else had a certifiable anchor extending b , in which case (a) applies to b . If b_j^* is a DA-certified base, it is certifiable and (a) applies directly. If b_j^* is an extension block, all $n - f$ votes endorse it, so at least $n - 2f \geq 2f + 1$ correct processors DA-voted its path, and the first dichotomy applies. The excluded case, $k_j^* = 0$ over a hash-anchor base, is exactly the one in which b_j^* is the previous safe-to-extend tip reaffirmed, a block that need not be certifiable, so that (a) does not apply. For the weaker claim, note that every block of $\text{Emit}(P)$ lies, on its chain, at or below either a finalised tip of the final segment or a coordinate of Tips for some V-QC on the reference chain; a rank-0 hash-anchored finalised tip equals the previous safe-to-extend tip, so in every case (b) applies. $\square$

So, the guarantee is as follows: quorum-scale redundancy ($2f + 1$ correct holders) for everything at or below finalised tips that certify new material, with a single guaranteed holder otherwise. A single correct holder suffices for retrieval, but the redundancy gap is worth quantifying. It concerns, per chain, only the blocks above the last certifiable block or certified base, and its *persistence* requires a faulty producer, whether equivocating or merely withholding, say by disclosing blocks to a single correct processor. On an honest chain every block is eventually DA-voted by all correct processors (Theorem 2, Step 1), hence certifiable, and (a) restores $2f + 1$ holders.

Membership finality. Order finality of extensions requires unanimity above a fully endorsed proposal (see the end of this subsection). A useful guarantee is nevertheless available at a much weaker threshold. Say a transaction block b is *membership-finalised* by the L-notarisation Q if b lies strictly above the base of Q 's chain proposal for b 's chain, and at least $3f + 1$ of Q 's votes endorse b .²⁸ Membership finality is deliberately decoupled from the finalised tips of Section 4: there is no requirement that $k_i^* = m_i$, nor that b lie at or below $\text{Tips}^*(Q)_i$. One direction does hold: every block strictly above the base and at or below $\text{Tips}^*(Q)_i$ is membership-finalised by Q , since at least $3f + 1$ votes report positions of at least k_i^* , and an extension tip is endorsed by all $n - f$. The converse fails, and usefully so: an extension block endorsed by $3f + 1$ votes but short of unanimity lies strictly above $\text{Tips}^*(Q)_i$, its membership settled even though its placement in the total ordering must await a later view.

Lemma 13 (Membership finality). *If b is membership-finalised by some L-notarisation, then no block incompatible with b is ever certifiable.*

Proof. Suppose b' , incompatible with b , were certifiable, diverging from b 's path at height h_0 , so that at least $n - 2f - f_a$ correct processors DA-voted b' 's path at h_0 . At least $3f + 1 - f_a$ correct processors' votes endorse b . Consider any one of them, p_j say. Since b lies strictly above

²⁸The first condition, that b lie strictly above the base, cannot be dropped. Since endorsement is closed under ancestry, every vote endorses the base and everything below it, without any voter having DA-voted those blocks, and a carried base needs as little as one correct DA-voter on its branch. Without the condition, an equivocating producer could have a block below the base 'membership-finalised' by all $n - f$ votes while $n - 2f$ correct processors certify an incompatible sibling.

the base, p_j 's endorsed tip does too, so p_j has genuinely DA-voted its endorsed path, from its certified anchor upward, and b lies on that path. If p_j 's anchor has height at least h_0 , the anchor is a certifiable block whose path agrees with b 's at h_0 , so is incompatible with b' , and Lemma 1 yields a contradiction with b' 's certifiability directly. Otherwise p_j DA-voted b 's path at h_0 . If the latter holds for all such p_j , then, correct processors DA-voting once per height, the two sets are disjoint, and $(3f + 1 - f_a) + (n - 2f - f_a) \leq n - f_a$, i.e. $f + 1 \leq f_a$, a contradiction. $\square$

The threshold is exact, uniformly in f_a . For fault count f_a , the proof shows $2f + f_a + 1$ endorsing votes suffice; and $2f + f_a$ do not: with f_a faulty processors, the correct processors can split as $2f$ DA-voting b 's branch and $n - 2f - f_a$ DA-voting b' 's, whereupon b gathers $2f + f_a$ endorsing votes in an L-notarisation while b' is certified. Since the protocol must be safe for every $f_a \leq f$ without knowing f_a , the threshold $3f + 1$ is forced, and optimal.

We should be precise about how much the lemma gives, since its conclusion is weaker than it may first appear. Directly, it constrains certification only: every block subsequently certified on the chain is compatible with b , so the chain's certified spine can never leave b 's branch. Since uncertified blocks may nevertheless be finalised, this neither places b in the ledger nor, on its own, excludes incompatible blocks from the ordering. Both upgrades arrive with larger counts, and we state them for the case that matters below, in which b 's producer is correct, so that blocks incompatible with b do not exist at all. When at least $n - 2f$ votes of *every* L-notarisation endorse b , which is what Theorem 3 below provides for well-circulated blocks of correct producers, any later finalisation of the chain at or beyond b 's height extends b , and places b with it. Whether the chain is finalised again at such heights is a liveness matter rather than a safety one. For a correct producer this is guaranteed, as in the proof of Theorem 2. A faulty producer, by contrast, may stall its own chain indefinitely, and not only by falling silent: after an equivocation, correct processors that spent their one DA-vote per height on the rival branch are permanently unable to vote for b 's path below the certified frontier, so that even when a correct holder of b leads and proposes it, b 's branch may be able to muster only $3f + 1 - f_a$ correct reported positions, short of the $3f + 1$ that finalisation requires unless faulty processors choose to assist. Consistent with damage localisation, the only chain that can be wedged in this way is the equivocator's own.

Block inclusion. We now spell out the censorship resistance obtained, which is a pay-off of extension voting. The key point to watch is which quantifiers depend on the leader: none do.

Theorem 3 (Block inclusion). *Let b be a block on chain i produced by the correct processor p_i , and let v be a view in which some leader block $\ell = (v, H(Q), C)$ receives an L-notarisation. Suppose that:*

- (i) *every correct processor DA-voted b , and each ancestor of b lying strictly above $\text{Tips}(Q)_i$, before sending its view v vote, and;*
- (ii) *b 's height exceeds that of $\text{Tips}(Q)_i$ by at most e .*

Then:

- (a) *the vote of every correct processor that votes for ℓ endorses b ;*
- (b) *at least $n - 2f \geq 3f + 1$ votes of every L-notarisation for view v endorse b ; in particular, whenever b lies strictly above the proposal's base, b is membership-finalised by every one of them (and when instead a DA-certified base lies at or above b , every certifiable block is compatible with b by Lemma 1 in any case);*

- (c) b is an ancestor of (or equal to) $\text{Tips}(W)_i$ for every view v V-notarisation W , and;
- (d) b appears in $\text{Emit}(P)$ for every pool P of $n - f$ or more votes for a leader block of any later view that receives an L-notarisation: b is ordered by the next finalised leader block.

Proof. Since p_i is correct and signatures cannot be forged, the blocks of chain i form a single path, one block per height.

(a) Let p_j be a correct processor voting for ℓ , with vote (π, ε) , and consider the base for chain i in C . By proposal validity (checked by the first correct voter, and inherited along the vote-along rule as in Lemma 6), the base extends or equals $\text{Tips}(Q)_i$. If b lies at or below the base, then b is an ancestor of every block p_j 's vote endorses on chain i , and we are done; likewise if the position block $\text{Prop}(C, i, \pi(i))$ lies at or above b on the chain's single path, since the endorsed tip extends the position block. Otherwise b lies above the base, and by (ii) at most e above $\text{Tips}(Q)_i$, hence at most e above the base (which extends $\text{Tips}(Q)_i$). The position block is a real block of the chain, DA-voted by p_j when $\pi(i) \geq 1$ and equal to the base otherwise, and it is an ancestor of b , since the real entries lie on the chain's single path below b . By hypothesis (i), p_j has DA-voted every block strictly between $\text{Tips}(Q)_i$ and b , so its DA-voted blocks run consecutively from the position block to b , and b lies within e of the position block. $\text{Extensions}(\mathcal{S}, \ell)$ therefore carries p_j 's endorsed tip on chain i to b or beyond, so p_j 's vote endorses b .

(b) An L-notarisation contains votes from $n - f$ distinct processors, at least $n - 2f$ of them correct. By (a), each such vote endorses b , and $n - 2f \geq 3f + 1$.

(c) Let W be a view v V-notarisation with tally V_i . Correct processors appearing in W appear with their actual votes. Since ℓ receives an L-notarisation, at least $n - 2f$ correct processors voted for ℓ , of which at most f are absent from W , so at least $n - 3f \geq 2f + 1$ votes in V_i endorse b . If b lies at or below the position block $\text{Prop}(C, i, k_i)$, then $b \preceq \text{Tips}(W)_i$ immediately. Otherwise b extends the position block (single path) and has $2f + 1$ endorsing votes in V_i , so b is a carry candidate and the carry applies. Candidates lie on the single path, and a deepest candidate has height at least b 's, so extends b : $\text{Tips}(W)_i$ extends b .

(d) Let ℓ' , of view $v' > v$, receive an L-notarisation, and let P be a pool of $n - f$ or more votes for it. By Lemma 6, ℓ is an ancestor of ℓ' , so the chain of references from ℓ' passes through a V-QC Q_v for view v , and $\text{Emit}(P)$ extends $\text{Ord}(Q_v)$. By (c), $\text{Tips}(Q_v)_i$ extends b . Finally, b appears in $\text{Ord}(Q_v)$: along Q_v 's own reference chain the coordinates $\text{Tips}(\cdot)_i$ are non-decreasing in height (each proposal's base extends or equals the previous safe-to-extend tip) and all lie on chain i 's single path, so the Horiz segments partition the heights below $\text{Tips}(Q_v)_i$, and b is appended in the segment covering its height. $\square$

One addition is available when the leader proposes honestly for chain i , and we state it separately because, unlike everything in the theorem, its hypotheses depend on the leader.

Corollary 2. *In the setting of Theorem 3, suppose additionally that b is compatible with the proposed tip for chain i .*

- (a) *If b lies at or below the proposed tip, then the finalised tip on chain i extracted from every L-notarisation for view v , and from every pool of $n - f$ or more view v votes, extends b .*
- (b) *If b extends the proposed tip, the same holds for every pool containing the votes of all correct processors (though not necessarily for every individual L-notarisation, whose unanimity f faulty votes can deny).*

In either case b is finalised within the view, although its placement in the total ordering may await a later view.

Proof. (a) If b lies at or below the base, then every extracted tip extends the base, which is position 0, and hence extends b . Otherwise write $b = \text{Prop}(C, i, m)$, for some $m \leq m_i$. The proposed entries below b are ancestors of b lying above the base, so by hypothesis (i) of the theorem every correct voter has DA-voted $\text{Prop}(C, i, k)$ for every $k \in [1, m]$, and reports $\pi(i) \geq m$. Every L-notarisation therefore contains at least $n - 2f \geq 3f + 1$ votes with $\pi(i) \geq m$, so its position rank k_i^* is at least m , and the same holds for the rank of any pool of $n - f$ or more votes. The extracted tip is $\text{Prop}(C, i, k_i^*)$ or a block extending it, and so extends b .

(b) If b extends the proposed tip, then *every* proposed entry is an ancestor of b lying above the base, so every correct processor that votes for ℓ reports $\pi(i) = m_i$, and by Theorem 3(a) its vote endorses b . A pool containing the votes of all correct processors thus contains at least $n - f$ votes with $\pi(i) = m_i$, giving position rank $k_i^* = m_i$, and at least $n - f$ votes endorsing b , so the extension step applies and b_i^* is a deepest block that $n - f$ of the pool's votes endorse. Any two such blocks are compatible, since their endorsing sets intersect in a correct processor, whose DA-votes determine a single path; a deepest one therefore extends b .

In either case, placement in the total ordering may await a later view, since emission halts at the empty slots of unsettled chains (Section 4). $\square$

Part (a)'s additional hypothesis is exactly what junk entries deny (Section 3.6): the one thing a leader can withhold is this in-view finalisation, never the membership and carry guarantees of the theorem. Part (b) makes formal the observation of Section 3 that the unanimity required for extension finality is unanimity within a *single* L-QC, and that an L-QC with the required unanimity can always be assembled from the pool once the correct votes have arrived, whatever the f faulty voters do.

What the theorem gives. Hypotheses (i) and (ii) mention only dissemination: (i) holds whenever b and its predecessors reach the correct processors before they vote (correct processors DA-vote eligible blocks immediately, so one δ of dissemination suffices), and (ii) is a freshness bound, satisfied by everything produced since the previous view's tips up to depth e . Neither hypothesis mentions the leader, and the conclusions quantify over *every* L-notarisation and *every* V-notarisation of the view. The view v leader (by junk entries, stale entries, a low anchor, equivocation, or any other behaviour) can influence which of these objects exist, but not what they contain regarding b . The consequence is best put as a dichotomy: *either view v produces no L-notarisation at all, or b is membership-finalised in view v and ordered by the next finalised leader block.* A Byzantine leader cannot have its block finalised *and* exclude a well-circulated honest block. The only censorship available is to sacrifice the view (producing nothing, harming every chain equally, and forfeiting a leader slot) and even this merely delays b by the one view, since hypothesis (ii) is then re-established with respect to the carried tips. Contrast Raptr's easy spoiling (Section 3.6), where a single faulty *producer* voids the optimistic path for the whole block. Here even the *leader* cannot void inclusion, and faulty producers can affect no chain but their own. What remains leader-sensitive is only in-view *order* finality on targeted chains (the one-view delay of Section 3.5, available to the leader alone, at the price of attributability).

We note the alternative we did not take: having processors echo leader blocks to one another before voting would make even in-view order finality leader-proof (voters could then verify that the entries they lack were genuinely withheld rather than never sent), at the cost of $O(n^2)$ communication in leader-block-sized objects and timeouts inflated by an additional δ . The dichotomy above is what makes the cheaper design defensible: what the leader can suppress is immediacy, never membership.

Chain-local finality. Finalisation fixes a chain’s contribution to the ledger up to the finalised tip; a later view settles only the cross-chain interleaving, together with the chain’s entries at greater heights. We make this explicit, since the naive reading fails in two instructive ways.

Corollary 3 (Chain-local finality). *Let the view v leader block $\ell = (v, H(Q'), C)$ receive an L -notarisation, let P be a pool of votes for ℓ by $n - f$ or more distinct processors, and let b_i^* be the finalised tip for chain i extracted from P . Then:*

- (a) *in the ordering extracted from any V-QC of view at least v , and in any emission at view v or later, every chain i block at height at most $b_i^*.height$ either appears in $\text{Ord}(Q')$ or is an ancestor of b_i^* , and;*
- (b) *chain i ’s blocks appear in any such ordering in strictly increasing height order.*

In particular, chain i ’s contribution to the eventual ordering at heights at most $b_i^.height$ is determined at finalisation, by $\text{Ord}(Q')$ and b_i^* : nothing at those heights is added, removed or reordered by later views.*

Proof. (b) holds by construction: each step of the recursion defining Ord appends, on chain i , a segment of strictly increasing heights lying strictly above the chain i coordinate of the referenced V-QC’s tips, and successive coordinates never decrease in height, since bases extend the tips they replace or are DA-certificates of greater height, and tips extend bases. For (a), consider first a V-QC W for view v itself. By Lemma 4 the leader block contained in W is ℓ , so W references Q' and $\text{Ord}(W) = \text{Ord}(Q') \frown \text{Horiz}(\text{Tips}(Q'), \text{Tips}(W))$, while by Lemma 7 $\text{Tips}(W)_i$ extends b_i^* . The chain i blocks appended beyond $\text{Ord}(Q')$ lie on the path through $\text{Tips}(W)_i$, so those at heights at most $b_i^*.height$ are ancestors of b_i^* . For a V-QC W of view $v' > v$, Lemma 6 places ℓ on the reference chain of the leader block contained in W , via some view v V-QC $Q^{(0)}$, so $\text{Ord}(W)$ extends $\text{Ord}(Q^{(0)})$, and every chain i block appended beyond $\text{Ord}(Q^{(0)})$ lies strictly above $\text{Tips}(Q^{(0)})_i$, which extends b_i^* , hence at height greater than $b_i^*.height$. Emissions are prefixes of such orderings, by Lemma 8 and the chaining in the proof of Theorem 1. $\square$

When the producer of chain i has not equivocated, the description simplifies. In this case, the blocks in question are exactly b_i^* and its ancestors, in height order, so the chain’s content through b_i^* is final in every sense but its interleaving with other chains. Applications whose state partitions by chain, per-producer app-chains, payment lanes or per-account sequencing, say, may therefore execute chain i through b_i^* at finalisation, and need never wait for placement in the total ordering, recovering the one-view wait exactly in the case that the sweep halts ahead of chain i ’s slots. Two qualifications make the interface exact. First, logs deduplicate transactions across chains, and which occurrence of a duplicated transaction survives depends on the interleaving, unknown at finalisation; the interface is exact for transactions bound to a single chain, and harmless whenever execution is idempotent under duplicates. Second, an equivocating producer’s entries *above* b_i^* may later land on a branch incompatible with blocks an executor holds, so an executor should suspend a chain upon proof of equivocation rather than speculate beyond the finalised tip. Early execution of this kind is the proactive complement of reordering by blame sets (Section 6): the one protects applications from the placement wait, the other shrinks the wait itself.

Accountability. The summary of Section 3.6 distinguishes leaders that avoid leaving evidence of misbehaviour, and it remains to say what that evidence can be. Misbehaviour in Multimitt

falls into three classes. *Provable* misbehaviour is constant-size, self-evident, and verifiable offline, consisting in each case of a pair of signed objects that no correct processor can produce: two transaction blocks signed by the same producer at the same height, or a signed block whose parent hash contradicts the producer’s signed block one height below (the header structure makes both checks header-only); two leader blocks signed by the same leader for the same view; two view v votes, or a view v vote and a $\text{novote}(v)$, signed by the same processor (Lemmas 2 and 3); and two DA-votes signed by the same processor for the same chain and height (threshold shares being individually attributable to their signers). Junk proposal entries are merely *attributable*: no one can prove that a proposed payload corresponds to no signed block, but a challenge to exhibit the signed header behind an entry is always answerable by a correct leader, which proposes only blocks it holds, and never answerable for junk, since the producer’s signature cannot be forged. Omission, finally, is *not even attributable*, since a leader that proposes less than it received is indistinguishable from a leader that received less, which is why the guarantees of Section 3.6 treat omission separately, its damage capped by extensions, rather than deterring it. We do not formalise the responses (slashing, reputation) that each class of evidence can support, and note only that a missing challenge answer is not proof in the offline sense, since under asynchrony a correct leader’s answer may simply be late.

The exact extension threshold. Finally, we justify the claim of Section 4 that the extraction rules cannot be relaxed at $n = 5f + 1$, and determine exactly what relaxation becomes available for larger n . Consider rules of the following form: an extension block is finalised on the strength of s votes endorsing it in an L-notarisation, and an extraction rule assigns safe-to-extend tips to each V-notarisation, as a function of the V-notarisation alone. Say the pair is *safe* if, in every execution, the tips assigned to every view v V-notarisation extend every extension block so finalised in view v . This is the contract that Lemma 7 discharges for the rules of Section 4, and on which consistency rests. Set $s^* := \lceil (n + 3f)/2 \rceil$.

Theorem 4 (Exact extension threshold). *For every $n \geq 5f + 1$:*

- (a) *no safe pair exists with $s < s^*$, and;*
- (b) *a safe pair exists with $s = s^*$.*

At $n = 5f + 1$ we have $s^ = n - f$, so the rules of Section 4 are optimal there.*

Proof. (a) We construct two executions sharing a view v V-notarisation W bit for bit, in each of which some extension block gathers $s^* - 1$ endorsing votes in an L-notarisation, the two blocks incompatible. No extraction rule is then safe in both executions, since identical bytes force identical tips, and no block extends both. The producer p_i of the chain in question is faulty in both executions. It signs incompatible extension blocks b and b' , at the same height above the proposed tip, showing b to some processors and b' to others. W consists of $n - f$ votes for the view’s leader block: $x := \lceil (n - f)/2 \rceil$ endorsing b , with p_i ’s own vote among them, and $y := \lfloor (n - f)/2 \rfloor$ endorsing b' . The f processors unaccounted in W are correct in both executions. In the first execution, the faulty processors are p_i together with $f - 1$ of the b' -side voters (there is room, since $y \geq 2f$). The producer shows b to the unaccounted processors, who endorse it in their votes, and each faulty b' -side voter signs a second vote endorsing b . The votes endorsing b then come from $x + f + (f - 1) = x + 2f - 1 = s^* - 1$ distinct processors, and an L-notarisation containing them all is assembled by filling the quorum with the remaining votes of W . The second execution is the mirror image, with faulty set p_i together with $f - 1$ of the

b -side voters, and one addition: p_i 's accounted vote endorses b , so its *second* signature, endorsing b' , now contributes a distinct processor, and b' gathers $y + f + (f - 1) + 1 = y + 2f \geq s^* - 1$ endorsing votes. Both executions are otherwise unremarkable, and W is a valid V-notarisation in each. Note the asymmetry: the producer's accounted vote sits on one side, and whichever side holds it is capped at $2f - 1$ additions, the other at $2f$; this single pinned vote is what separates $s^* - 1$ from s^* .

(b) Take $s = s^*$, and extract as in Section 4 except that the extension carry runs at the threshold $s^* - 2f$ in place of $2f + 1$, with one further rule: when two incompatible blocks each gather exactly $s^* - 2f$ accounted endorsing votes, which we show below is possible only when $n + 3f$ is even, discard the one that the producer's own accounted vote endorses. First note that ties are the only ambiguity. Votes endorse a single path, so the endorsement counts of pairwise incompatible blocks sum to at most $n - f$, while two blocks meeting the threshold account for $2(s^* - 2f) \geq n - f$, with equality only when $n + 3f$ is even; three are excluded outright, and in the odd case so are two.

Now suppose the extension block b is finalised, with s^* endorsing votes in some L-notarisation. Let W be any view v V-notarisation, in an execution with $f_a \leq f$ faulty processors. At most f_a of the s^* endorsers are faulty, so at least $s^* - f_a$ are correct, and each endorses b 's path in its single vote. At most f of these are unaccounted in W . At least $s^* - 2f$ votes accounted in W therefore endorse b , and the carry reaches b .

Now let b' be any block incompatible with b . A correct voter endorses at most one of the two, so at most $(n - f_a) - (s^* - f_a) = n - s^*$ correct voters endorse b' , and b' 's accounted endorsement is at most $n - s^* + f_a \leq s^* - 2f$. If $n + 3f$ is odd, the inequality is strict, b dominates every rival, and we are done. So suppose $n + 3f$ is even and b' ties with b at $s^* - 2f = (n - f)/2$. The tie forces every inequality above to be an equality. So $f_a = f$, every faulty processor is accounted in W with a vote endorsing b' , the f unaccounted processors are correct endorsers of b , and the $(n - f)/2$ accounted votes endorsing b all come from correct processors. Each branch now carries at least $f + 1$ accounted correct votes (b' carries $(n - f)/2 - f = (n - 3f)/2 \geq f + 1$, since $n + 3f$ even gives $n \neq 5f + 1$, so $n \geq 5f + 2$). Blocks on both branches were therefore genuinely signed, so the producer equivocated and is faulty. The producer is accounted, the unaccounted processors being correct, so its vote is one of the f faulty accounted votes, and every one of those endorses b' . The override therefore discards b' and carries b , as required. Finally, on an honest producer's chain incompatible blocks do not exist, so no tie can arise and the override never fires. $\square$

Three remarks. First, at $n = 5f + 1$ the theorem's thresholds are $s^* = n - f$ and $s^* - 2f = 2f + 1$, no tie is possible ($n + 3f$ being odd), and the safe pair of part (b) is precisely the pair of rules of Section 4. Second, the theorem concerns the extension carry alone. The companion observation that, even at $s = n - f$, the condition $k_i^* = m_i$ cannot be dropped, by the two-execution construction described with the definition of Tips^* in Section 4, is unaffected, as are the downstream constants: membership finality at $3f + 1$ and the counts of the block inclusion theorem do not depend on s . Third, we do not install the relaxation. At the headline assumption $n = 5f + 1$ it changes nothing, and the rules of Section 4 stand as written; a deployment with n well above $5f + 1$ may adopt the generalised carry as a contained change. At $n = 100$, $f = 19$, it buys an in-view ordering of the frontier that survives two omissions from the quorum's 81 votes, where unanimity tolerated none.

5.4 Communication complexity

A basic feature of the protocol is that consensus traffic is independent of throughput. Transaction data travels once, on the chain layer. Consensus objects carry hashes and positions, at the sizes of Section 3.7. So long as bandwidth handles the consensus traffic with room to spare, the remainder of each link carries payload. The operative question is therefore not asymptotic but concrete: Are consensus objects large enough to intrude at the deployment scales one expects in practice? At $n = 50$ or 100 they are not, per the numbers of Section 3.7. At much larger n , correspondingly more bandwidth must be provisioned.

Here, we describe the asymptotics per view, taking hash and signature sizes to be constant, along with d and e . Message complexity is $\Theta(n^2)$. Votes travel all-to-all, which is the price of finalisation everywhere within 2δ of the proposal. In bits, a vote is $O(K)$. The vote exchange therefore costs $\Theta(Kn^2)$ in total, and $\Theta(Kn)$ per processor per view in each direction, with proposals adding $O(Kn)$. The one term that can exceed this is the dissemination of V-QCs. The forwarding duty of Section 4, taken literally, sends up to n^2 copies of objects that degrade to $\Theta(nK)$ (Section 3.7). This means $\Theta(n^3K)$ bits per view, and $\Theta(n^4)$ when $K = \Theta(n)$. The worst case is an artifact of unconditional forwarding, however. Votes and novotes already travel all-to-all, so in the common case every processor assembles each V-QC locally, and forwarded bodies are redundant. Simple modifications therefore suffice to remove the all-to-all forwarding of V-QCs:²⁹

- The forwarding duty is retained for nullifications, which are constant-size, and dropped for V-QCs.
- V-QCs travel by *targeted push* instead. A processor sends each first-new V-QC for a view v to the leader of view $v + 1$. A processor that votes for a proposal it has checked valid also sends that proposal’s anchor to the leader of the next view. Each rule costs at most one object per processor per view.
- The leader disseminates its anchor (the V-QC referenced by its proposal) together with its leader block.
- The view-entry rule is generalised to accept an exit object for any view at or above the current one, so a single fresh object suffices for entry from any depth. A constant-size status message on view entry, with pull-based catch-up extending the retrieval convention above, covers the residual case of a processor unable to assemble an exit object locally.

Per-view communication is then $O(Kn^2)$ bits in *all* regimes, i.e. $O(n^3)$ when $K = \Theta(n)$, matching the vote exchange itself. We have verified the modified protocol in full, re-proving each affected lemma of this section, and omit the proofs to conserve space. The liveness analysis carries over, with timeouts unchanged.

6 Optimisations

Backup proposals: finalising the views of crashed leaders. During synchrony, the one case in which a view finalises nothing however the non-leader processors behave is a failed leader.

²⁹One might ask why the specification of Section 4 does not simply adopt these modifications. The reason is that the all-to-all forwarding costs bandwidth rather than latency. As noted with the forwarding rule of Section 4, nothing in the protocol waits on the redundant sends, which lie off the critical path, and at the scales of Section 3.7 the bandwidth involved is small. Keeping the unconditional duty keeps the specification and its analysis simpler.

In this case, correct processors time out, the view nullifies, and every chain waits a view. The wait is short, and Section 3.6 argues that the case is rare when leaders have something to lose, but it is removable, at the price of one further proposal per view. Proposer redundancy has a long history: PBFT’s view change is a backup leader across views, RBFT [8] runs redundant leaders in parallel to police a faulty master, and DAG protocols place several anchors per round so that one crashed leader costs no timeout [70, 9] (Section 8). The immediate inspiration is Hermes [60], concurrent work in which every processor proposes at view entry, so that even the view of a failed leader finalises. Our design is narrower: one designated backup rather than a proposal from every processor, dealing with crashed leaders only and making no attempt at the tail-latency guarantees that Hermes targets. To do so, we give each view a *backup*, $\text{backup}(v) := \text{lead}(v + 1)$ (any fixed assignment with $\text{backup}(v) \neq \text{lead}(v)$ suffices). Leader blocks gain a role bit: $\ell = (v, r, q, C)$ with $r \in \{0, 1\}$, included in the identifier, $H(\ell) := H(v, r, q, H(C))$, and signed by $\text{lead}(v)$ when $r = 0$ (*the leader’s proposal*) and by $\text{backup}(v)$ when $r = 1$ (*the backup proposal*); condition (i) of proposal validity is checked per role, and backup proposals are only valid upon timeout. The backup proposes upon entering the view, exactly as the leader does and via the same **SelectAnchor**, the two proposers choosing anchors independently. Voting changes only at the timeout (before its own timeout, a processor votes for the backup proposal only via the vote-along rule, upon a V-QC designating it): a processor reaching 2Δ on its timer without having voted or nullified votes for the backup proposal if it holds a valid one, and sends its novote–nullify pair only otherwise. Everything else is unchanged.

This modification suffices to deal with leaders that crash before view entry. If $\text{lead}(v)$ crashes and $\text{backup}(v)$ is correct, the backup proposal has been disseminated upon view entry, every correct processor votes for it at its timeout, and the votes assemble into an ordinary L-QC. The view then obtains the finalisation and extraction guarantees of a correct-leader view, extension votes included, by $t + 2\Delta + 2\delta$ from the first correct entry, which is the time at which the rules of Section 4 merely assemble the nullification, deferring all content by a view. One way to put the gain: under crash faults arriving independently of the leader schedule, the extension roughly *squares* the probability that a view is lost to silence, since failure now requires both of the view’s proposers to be down; a view-failure rate of one in a hundred becomes roughly one in ten thousand. The good and common cases are untouched. In those cases, the timeout never fires, correct processors send exactly the messages of Section 4, and the backup proposal is the only extra message sent. Safety reduces to familiar counting checks. The counting behind the designation lemma ensures that, if any view v block receives an L-notarisation, then no other view v block receives $2f + 1$ votes. If any proposal is L-notarised, the designation and no-nullification lemmas of Section 5 hold for it verbatim, their proofs nowhere using the author’s role: every V-notarisation of the view designates the finalised proposal, and the view cannot be nullified. In a view where nothing is L-notarised, distinct view v blocks (of the two roles or, under equivocation, versions within a role) may separately gather $2f + 1$ votes, so distinct V-QCs may designate distinct proposals; the leader-chain induction simply applies to every block receiving $2f + 1$ votes, its earliest-correct-voter argument gaining a third route, the timeout vote, which like the leader route presupposes a valid proposal. With these three observations in place, each proof of Section 5 applies with ‘leader block’ ranging over both roles.

A Byzantine leader can still spoil the view by *splitting* correct voters between the two proposals so that neither reaches $n - f$; the view then advances without finalising, by a designated V-QC for one of the proposals or by a nullification assembled via the evidence route, exactly as a failed view does in the base protocol. What a split view retains is membership finality: the $3f + 1$

threshold of Section 5 applies to the pooled votes across the view’s proposals, a vote counted once the proposal body it names is held, and the proof indifferent to which proposal each vote names (each vote’s side condition is checked against the base of the proposal it names).

The guarantees against censorship are improved. The block inclusion theorem quantifies over every L-notarisation of the view, now of either role, so a censoring *backup* faces the same dichotomy as a censoring leader. A leader that sacrifices its view by silence no longer even ensures the one-view delay, since a correct backup completes the view with the censored block included; to retain the delay, the leader must instead split correct voters by disseminating a valid proposal selectively. The backup slot itself is inert in good views: backup votes exist only where the leader has already failed for the voter concerned, a backup cannot bootstrap votes for its own proposal (a V-QC designating it presupposes $f + 1$ correct timeout votes), and anchoring artificially low is excluded for backups exactly as for leaders. The costs are one proposal-sized object per view, one bit in the leader-block identifier, and, in views where the timeout route is exercised, V-QCs recording votes for the non-designated proposal in the slot that Section 4 already provides for votes under an equivocating leader.

Skipping leaders that appear crashed. Although the backup proposal removes the cost of a crashed leader, a cheaper measure can also be used to shorten the induced delay. On entering a view, a processor checks whether it has heard anything from the leader within some period of time (to be set as a parameter). If it has not, while enough other processors have been active to rule out its own disconnection, the processor treats its timer for the view as already expired, and acts upon entry exactly as it would at 2Δ on its timer. Safety is unaffected, since the analysis of Section 5 nowhere constrains how timeouts are chosen. Liveness asks only that the rule eventually stop firing on correct leaders, which holds for any period long enough that every correct processor is heard from within it once the network stabilises.

Ordering variants: reputation without fragility. Multimmit is designed to be robust *without* a reputation mechanism, and reputation mechanisms bring complexities of their own (Section 3.6). Still, there are circumstances in which such a system may be useful in minimising the one cost that faulty producers still impose on other chains: the one-view placement wait, incurred when emission halts at an unsettled chain’s slot. The horizontal ordering of Section 4 can be replaced. An ordering that visits likely laggards last reduces the placement wait directly. Concretely, let processors maintain a *blame set* of at most f chains, by any rule on which all correct processors agree. The replacement sweep visits the positions of unblamed chains first, in horizontal order but with the chain order shuffled each view by a VRF seed on the view number, so that no chain index enjoys a standing positional advantage.³⁰ It then visits the blamed chains vertically, one chain at a time in the same shuffled order, so that a halt at one blamed chain defers only the chains behind it.

This sweep suffices, whatever the blame rule. It changes only the order in which positions are visited. That order is fixed per view by data held by every processor computing the view’s segment, namely the agreed blame set and the seed, which travels with the leader block, and it does not depend on the tips being swept towards. Each chain’s positions are still visited in increasing height, and the rule that appends, skips a settled chain, or halts is untouched. The arguments of Section 5 use nothing more. The prefix lemma and the monotonicity of emission

³⁰A hash of the leader block would serve as a seed, but is grindable by the leader; the VRF admits one value per view.

compare sweeps position by position in the common order, and block inclusion and liveness use only which blocks a segment contains, which reordering never changes. So consistency, finality and inclusion are unaffected, and a misblamed honest chain loses only its position in the sweep, never in-view emission, since a settled chain is emitted wherever the sweep places it.

We do not specify a blame rule, since attribution is subtle. A chain may show short because its producer withheld blocks, because the *leader* proposed junk entries, or through passing asynchrony; separating the first two cases requires the challenges of the accountability paragraph of Section 5. We note only that a sound rule must track the rank- $(3f + 1)$ statistic at which placement halts, not the rank- $(f + 1)$ statistic at which safe-to-extend tips advance: a producer disclosing each block to exactly $2f + 1$ correct processors keeps its tips advancing in every view, while halting every correct processor's emission at its slot.

Ordering variants: the proposal before the extensions. A second variant changes only the sweep. It visits the positions at heights up to each chain's proposed tip first, in horizontal order, and only then the positions beyond, again in horizontal order. Under the ordering of Section 4, disagreement among extension votes on one chain can delay the placement of other chains' content in the total ordering, proposal entries included, since the halt at the lagging chain's slot defers every position behind it. Under the split sweep, a halt in the extension phase defers extension content only. Within each view, extension votes then act purely as a bonus. They can add content beyond the proposed tips, and can no longer hold anything else back. The checks above apply verbatim.

Optimising the anchor choice. `SelectAnchor` returns the first V-QC the leader received for the greatest view held, an object already forwarded to all, so the anchor costs no further dissemination. However, this approach to V-QC selection is policy, rather than a required feature of the protocol. Validity accepts any V-QC for the referenced view, and the analysis of Section 5 quantifies over arbitrary V-QCs, as it must, since a Byzantine leader is free to anchor on anything it can assemble. Subject to the greatest view held, which liveness uses, the choice is free, at the price of one send: a leader anchoring elsewhere must disseminate its chosen anchor itself. The freedom is best spent on higher tips. Everything at or below the anchor's tips is placed in the total ordering by the never-halting recursion defining `Ord`; only the territory above is subject to the one-view placement wait. Since voters send their votes to all, the leader of a later view holds the previous view's vote *pool*, and may assemble an anchor from it: packing the tally with every vote it holds, and completing the count of $n - f$ distinct signers, where the votes alone fall short, with novote messages it has received. (Received V-QCs cannot be remixed, their signatures being aggregated; the candidates are whole held V-QCs and quorums assembled from individually held messages.) Exact tip-maximisation is combinatorial once a producer has equivocated, but nothing requires it: any valid choice is safe. The benefit is nil in the good case, and bounded by $d + e$ blocks per chain otherwise.

Nothing, moreover, requires an assembled V-QC to account for exactly $n - f$ messages: the leader may include *every* view v' message it holds. The counting arguments of Section 5 bound absent correct voters from above and tip endorsement from below, and both bounds improve as messages are added; position ranks are monotone under added votes, and any $f + 1$ endorsers include a correct processor, so availability is undisturbed. One point, however, requires care. With exactly $n - f$ messages the extension carry is unique, two incompatible candidates needing $2(2f + 1)$ endorsers among $4f + 1$ messages. With more messages, incompatible candidates can

each reach $2f + 1$, and the carry needs a deterministic tie-break. Any tie-break is safe: a block finalisable by any L-QC or pool leaves at most $2f$ votes, in any message set whatever, endorsing blocks incompatible with it, so competition arises only when nothing conflicting is finalisable. The pool rules for L-QCs already take this many-message form (Section 4). The anchor-side generalisation is their mirror image.

Tip commitments: verifiable history without its V-QCs. Leader blocks hash-chain all of history through their V-QC references, so the trust root is already of constant size, but the data is not. $\text{Tips}(Q)$ is computable from Q , but a claimed value of $\text{Tips}(Q)$ cannot be checked without Q , so the V-QCs are the only witnesses to the ordering: replaying the ordering from a trusted leader block means fetching every V-QC of the reference chain and computing the extraction of Section 4 view by view, at tens of kilobytes per view in the good case (Section 3.7), more when faulty voters inflate deviation records, and the cost is paid by every processor that ever syncs. One extra field can be used to remove the cost. Let leader blocks take the form $\ell = (v, H(Q), C, c)$, with validity requiring $c = H(c', T(Q))$, where c' is the commitment carried by the leader block contained in Q , and $T(Q)$ is the vector of pairs $(H(\text{Tips}(Q)_i), \text{Tips}(Q)_i.\text{height})$. The condition is checkable from ℓ and Q alone, since computing $\text{Tips}(Q)$ requires only Q . Note also that no leader block violating the condition is ever certified, by the argument of the data availability paragraph of Section 5: among the $2f + 1$ votes that certification requires, the first $f + 1$ correct voters vote via the valid-proposal route, and so check the condition. A single finalised leader block then authenticates the entire history of safe-to-extend tips, at approximately 40 bytes per chain per view, a hash and a height each (4KB per view at $K = 100$). Since each tip hash pins its ancestor path, the tips sequence together with the transaction blocks themselves determines the total ordering exactly, and historical V-QCs need never be touched. The retrieval convention of Section 5 remains, but is required only for live emission near the tip. The costs are one hash per leader block and per validity check, together with the loss of the observation that leader-block wire format matches Minimmit's.

One signature scheme. The setup of Section 2 equips processors with an aggregate scheme for votes and with two threshold schemes, at thresholds $n - 2f$ for DA-certificates and $2f + 1$ for nullifications. The threshold schemes can be dispensed with. A DA-certificate becomes $(\text{hd}(b), W, \sigma)$, where W is a bitmap of $n - 2f$ signers and σ aggregates ordinary signatures on $(\text{da}, \text{hd}(b))$, and nullifications are treated likewise. The message flow is unchanged, and no use the analysis makes of a DA-certificate consumes the threshold property: availability arguments count signers (and the bitmap tells a retriever whom to ask), uniqueness of certified blocks is quorum intersection over signer sets, and anchors and the DA window compare header values. Verification still costs one pairing, since the messages are common, together with a popcount that the scheme no longer enforces. The losses are modest. Certificates grow by $n/8$ bytes (13 bytes at $n = 100$), and are no longer unique as bit strings, so deduplication must key on content; the two places where the analysis invokes uniqueness of threshold signatures (deduplication, and the observation in Lemma 11 that what was forwarded is the nullification itself) instead read ‘the first nullification received or assembled’, mirroring the treatment of V-QCs. The gains are that the cryptographic model shrinks to the aggregate scheme the protocol already requires, that no distributed key generation is needed, and that a membership change becomes an update to a key list rather than a resharing ceremony. One consideration cuts the other way. A threshold certificate verifies against a single fixed group key, while the aggregate requires the validator

key list, so an external consumer that verifies DA-certificates and nothing else, a minimal light client, say, is arguably better served by the threshold instantiation. The choice is a matter of deployment, and nothing in the analysis depends on it.

Batch verification. Little here is specific to Multimmit, but the impact on performance is significant for the values of n considered in Section 3.7, since steady-state CPU cost is dominated by signature verification. A producer collecting DA-vote shares may aggregate optimistically and verify the combined signature once, falling back to per-share checks only on failure, and the same applies to nullify shares. A processor receiving a burst of votes for one view may verify them as a single multi-pairing under random scaling coefficients, bisecting on failure; and QC verification, costing one pairing per distinct message in the aggregate, batches across QCs the same way. In each case the optimistic path costs one batched pairing-product check (Miller-loop work still scaling with the number of distinct messages), and the fallback isolates the invalid objects, with individual attribution available for isolated vote signatures and threshold shares, an invalid signature so isolated being itself evidence of misbehaviour (Section 5). Transport offers similar headroom: votes may travel content-addressed, a digest standing in for a body the receiver can reconstruct, and the forwarding duty may be advertise-and-pull, bringing good-case consensus bandwidth close to the object sizes themselves; we leave the mechanics to the experimental companion.

7 Experiments

We compare Multimmit against BlueBottle-Core [78] and Raptr [75] on two testbeds, one global and one North American. Three findings are worth emphasising:

1. Multimmit’s latency is essentially flat as the incoming transaction rate rises from 50,000 to 1,000,000 transactions per second. The median moves from 388.5 to 400.0 ms on the global testbed and from 51.7 to 62.6 ms on the North American one.
2. This performance is robust against faults. With nine of fifty validators crashed, and separately with 0.1% message loss, Multimmit degrades gracefully. BlueBottle slows by orders of magnitude or fails to keep up with the incoming rate, and nine crashes slow Raptr by roughly an order of magnitude.
3. Multimmit’s consensus traffic is a small fraction of the traffic that carries the block bodies, a few percent at the higher transaction rates.

7.1 Setup and methodology

Table 4 summarises the testbeds, the protocol configurations, and the workload. Each testbed has 50 validators, each on an AWS `c8g.12xlarge` instance with 48 vCPUs, 96 GiB RAM, and a 1,000 GiB io2 volume provisioned for 16,000 IOPS. The global testbed spans 13 regions across North America, Europe, Asia, and Australia, with three or four validators per region. The North American testbed has 17 validators in Northern Virginia, 17 in Ohio, and 16 in Canada. The global topology matches the one used in BlueBottle’s published evaluation. All three implementations use the same instance types and storage provisioning.

	Global	North America
Validators	50	50
Regions	13, on four continents	N. Virginia, Ohio, Canada
Validators per region	3–4	16–17
Instance	AWS c8g.12xlarge, 48 vCPUs, 96 GiB RAM	
Storage	1,000 GiB io2, 16,000 provisioned IOPS	
Multimmit	$K = 50$ chains, $d = 32$, $e = 8$, $f = 9$, quorum 41, 2 s view timeout	
BlueBottle	artifact <code>e02aeba213fa</code> , two leaders per round, quorum 41, 1 s timeout	
Raptr	$f = 16$, quorum 34, eight subblocks, 200 ms QC-vote timer	
Transactions	synthetic 512-byte payloads, no execution	
Blocks	Multimmit batches 512 transactions into 256 KiB producer blocks	
Offered rates	50k, 100k, 250k, 500k, 1M tx/s. BlueBottle and Raptr through 500k	
Scenarios	healthy, 1 crash, 9 crashes, 0.1% message loss	
Schedule	120 s warmup, 120 s measured cohort, up to 120 s drain	

Table 4: Testbeds, protocol configuration, and workload.

Multimmit uses one producer chain per validator, $d = 32$, $e = 8$, $f = 9$, and a quorum of 41.³¹ Its view timeout is two seconds. Processors also apply the rule of Section 6 for skipping leaders that appear crashed. Here the rule fires when nothing has been heard from the leader for ten seconds while at least 41 validators have been active in that interval. The timeout is rarely exercised in these runs. Healthy views would be unimpacted by much smaller timeouts, and views led by stopped validators nullify on entry via the skipping rule, so a shorter timeout would leave the results unchanged. BlueBottle uses the published artifact at `e02aeba213fa`, with benchmark patches.³² The published artifact provides neither our workload nor our measurement points, and it has no controls for message loss or fsync. The patches add these and nothing else. They do not change consensus or block retention.³³ BlueBottle uses two leaders per round, a one-second leader timeout, and a quorum of 41. It makes the same $n \geq 5f + 1$ resilience assumption as Multimmit, so the two systems run with the same fault budget and the same quorum. Its WAL fsync is disabled throughout. Multimmit’s disk sync barriers are likewise disabled in this section. Appendix A measures the cost of enabling them.

Raptr uses the published implementation at `a4c98047cf8a`, with benchmark instrumentation and a minimal payload-retention fix.³⁴ Raptr uses $f = 16$ and a consensus quorum of 34, eight subblocks, and a QC-vote timer of 200 ms.³⁵ These are its native parameters under its $n \geq 3f + 1$

³¹Multimmit’s measurement base is `b9c8b515`, available at <https://github.com/commonwarexyz/monorepo/tree/cl/multimmit-paper-measurement>. Benchmark patches add workload instrumentation and message loss. For fsync-disabled runs, a further patch to `commonware-runtime` bypasses file, directory, and per-write sync barriers while retaining filesystem writes. Full revisions, configurations, and binary hashes are recorded in the released dataset.

³²BlueBottle’s benchmark source is revision `f49ed07b`, available at <https://github.com/clabby/mysticeti/commits/measure>.

³³We enable BlueBottle’s consensus-only mode through both of its switches, “`consensus_only`” and “`CONSENSUS_ONLY`”.

³⁴Raptr’s benchmark source is available at <https://github.com/clabby/aptos-core/tree/measure>, revision `233a90f9`. The patches add the scheduled workload, finality timestamps, traffic counters, message-loss injection, and fsync controls. They also adjust batching and storage limits and prevent historical payloads from retaining fetched transaction bodies. These changes do not alter the consensus protocol. The measured source and configuration revisions, together with binary hashes, are recorded in the dataset.

³⁵The QC-vote timer bounds how long a validator waits for missing batch data before voting for the longest

resilience assumption. Its quorum is smaller than the other two systems’ quorum of 41, which, if anything, favours its latency. The nine-crash experiment uses the same stopped validators across implementations, but does not exhaust Raptr’s larger fault budget. Raptr forms batches of up to 450 transactions and pulls up to 2 250 transactions, with a sender limit of five batches. Its Quorum Store quotas per validator are 300 MB in memory and 1 GB on disk. The receiver limit is 2 250 transactions and the dynamic rate limit is 15k transactions/s per validator. A preload shim bypasses `fsync` and `fdatasync` while retaining filesystem writes. All Raptr results use this shim.

Transactions are synthetic 512-byte payloads, with no execution. Measuring consensus without execution is standard in this literature.³⁶ The evaluations of BlueBottle [78] and Raptr [75] both stop their clocks before execution. We report throughput in transactions per second (tx/s). Multimitt admits transactions in batches of 512, which fill one 256 KiB producer block. BlueBottle receives submissions every 100 ms and retains its own block formation. Raptr also submits every 100 ms and retains its own batch formation. The offered rate is the incoming transaction rate we schedule for submission. Transactions are scheduled on a fixed timetable, divided among the surviving producers, and the timetable does not slow when a system falls behind. Offered rates are 50k, 100k, 250k, 500k, and 1M tx/s for Multimitt, and the same rates through 500k for BlueBottle.³⁷

Raptr uses the same offered rates through 500k. Of its 32 conditions, 22 pass the acceptance checks and ten are excluded for overload. These exclusions cover all global conditions at 250k and 500k, and North American operation with nine crashes at those rates. The global fault conditions at these rates were not run after the healthy conditions overloaded.

At each rate we test healthy operation, one crash, nine crashes, and 0.1% message loss without crashes. Crashed validators remain crashed and all three implementations crash the same validators. These are one validator in Northern Virginia for the one-crash case, and one in each of nine regions globally or three in each North American region for the nine-crash case. Loss is sampled independently for complete application messages at the receiver, after TCP delivery and before protocol processing. Discovery and connection-maintenance messages are exempt. TCP therefore does not repair the injected loss.

Each experiment at a given incoming transaction rate runs a 120-second warmup, followed by 120 seconds of scheduled submissions. We call the transactions scheduled in that window the cohort for the rate. When the window closes, some of the cohort has not yet reached finality. The run therefore continues without new input for up to another 120 seconds, so that these transactions can finish and be measured. We call this final period the *drain*. Faulty Multimitt runs end the drain early once the whole cohort has been admitted and finalised, subject to a ten-second minimum and passing health checks. Other runs use the full 120 seconds. We clear Multimitt’s data before each ascending sweep of rates. BlueBottle and Raptr start with fresh processes and data at every rate.

Latency starts at scheduled submission, before admission, transaction block construction, or

prefix it holds (this is the parameter written $\epsilon\Delta$ in the Raptr paper).

³⁶Execution time depends on the application and on parts of an implementation that lie outside consensus, so including it would blur comparisons.

³⁷BlueBottle exhausted memory at an offered rate of 1M tx/s on the global testbed with fresh processes and consensus-only operation. We therefore omit 1M from its measurements. The benchmark build retains the published consensus and block-retention code, so the memory use is that of the published implementation. The failure shows that this implementation cannot handle the rate with 96 GiB of RAM. It does not show that the protocol has a throughput limit there. An implementation that manages memory differently, or a machine with more of it, might sustain the rate.

signing, and ends at first local consensus finality at the transaction block producer (both timestamps use that processor’s clock). Raptr records consensus finality before fetching transaction bodies. The latency excludes ordered application delivery and execution. We count each transaction once and weight latency statistics by transaction count. Achieved throughput counts unique transactions finalised during the 120-second measurement window, including those submitted before it. The figures plot median latency with bars spanning P25 to P75, together with P99, against achieved throughput. Both axes of the latency figures are logarithmic.

A plotted observation must complete its cohort, sustain at least 95% of the offered rate, retain the expected live validators, and pass checks for input backlog, accounting, clocks, and telemetry. Wide latency distributions are retained when these checks pass. Incomplete cohorts have no latency point, since excluding unfinished transactions would bias the result. Each point represents one observed cohort, not an estimate of run-to-run variability. Every run is recorded in a machine-readable dataset released with the paper, including the failed runs the figures omit, with full statistics and validation records for each.³⁸

7.2 Latency

Table 5 and Figures 5 and 6 display the headline results of our experiments.

	Multimmit		BlueBottle		Raptr	
	median	P99	median	P99	median	P99
<i>Global testbed</i>						
Healthy, 100k tx/s	375.5	552.1	595.1	1 879.4	538.7	909.3
Healthy, 500k tx/s	384.8	669.0	2 652.0	13 377.2	overload	
Healthy, 1M tx/s	400.0	635.2	out of memory		not offered	
Nine crashes, 500k tx/s	613.4	1 397.5	below quorum		overload	
0.1% loss, 100k tx/s	388.4	556.1	21 366.9	60 085.2	545.2	918.9
<i>North American testbed</i>						
Healthy, 100k tx/s	50.6	65.3	120.7	1 868.2	147.9	227.6
Healthy, 500k tx/s	51.6	76.2	279.1	1 554.4	243.5	367.5
Healthy, 1M tx/s	62.6	99.8	not offered		not offered	
Nine crashes, 500k tx/s	103.7	503.2	999.3	4 769.0	overload	
0.1% loss, 100k tx/s	48.7	66.2	20 850.0	113 868.2	148.9	233.4

Table 5: Headline latencies in milliseconds, with fsync disabled. Entries marked out of memory or below quorum correspond to the failed runs described in the text. Rates above 500k were not offered to BlueBottle or Raptr. Raptr overload entries include conditions excluded after healthy operation overloaded, as described in the text.

Healthy operation, global. The comparison with BlueBottle is closest at lower transaction rates. At 50k, BlueBottle’s median is 596.8ms against Multimmit’s 388.5ms, and at 100k the

³⁸<https://commonware.xyz/artifacts/multimmit-measurements.json.zst>

figures are 595.1 and 375.5 ms. However, even at these lower transaction rates, the tails differ. BlueBottle’s P99 at 100k is 1 879.4 ms against Multimmit’s 552.1 ms. From 250k the medians differ more strongly. BlueBottle reaches 898.1 ms at 250k and 2 652.0 ms at 500k, where its P99 is 13 377.2 ms, and it exhausts memory at 1M. Multimmit’s median stays between 363.9 and 400.0 ms at every rate through 1M, and its P99 never exceeds 669.0 ms. At the two rates Raptr sustains, its latencies land between the other two systems. Its medians are 539.9 and 538.7 ms at 50k and 100k, with P99 values of 905.7 and 909.3 ms, a far shorter tail than BlueBottle’s. Raptr’s 250k and 500k conditions are excluded because overload prevents sustained throughput, with an incomplete cohort at the highest rate. These failures bound the tested implementation on these hosts, rather than establish a protocol limit.³⁹

Healthy operation, North America. The pattern for Multimmit and BlueBottle repeats at smaller scales. At 50k, the medians are 101.9 ms for BlueBottle and 51.7 ms for Multimmit, and BlueBottle’s P99 is already 1 626.5 ms against Multimmit’s 64.5 ms. At 500k the medians are 279.1 and 51.6 ms, with P99 values of 1 554.4 and 76.2 ms. At 1M, Multimmit’s median is 62.6 ms. Raptr sustains every healthy rate through 500k here. Its median rises from 145.9 ms at 50k to 243.5 ms at 500k, with P99 values of 228.3 and 367.5 ms, so at 500k it sits just below BlueBottle’s median and several times above Multimmit’s.

Nine crashes. On the global testbed, Multimmit’s median at 500k is 613.4 ms with a P99 of 1 397.5 ms. BlueBottle’s median reaches 6 990.9 ms at 250k. At 500k it exhausts memory and loses additional validators, leaving fewer live nodes than its quorum requires. Nine crashes cost Raptr more than either other system. Its global median at 100k is 7 834.7 ms and its P99 is 10 730.3 ms. This wider distribution passes the cohort and health checks. In North America at 500k, the medians are 999.3 ms for BlueBottle and 103.7 ms for Multimmit, with P99 values of 4 769.0 and 503.2 ms. Raptr’s median at 100k is 1 258.4 ms with a P99 of 2 596.7 ms. Its 250k and 500k conditions are excluded because overload prevents sustained throughput, with an incomplete cohort at the highest rate.

Message loss. Multimmit is essentially unaffected on either testbed. Its median at 100k is 388.4 ms globally and 48.7 ms in North America, in line with healthy operation. Raptr stays close to its healthy figures too. Its global median at 100k is 545.2 ms with a P99 of 918.9 ms, and in North America at 500k the median is 244.2 ms with a P99 of 366.1 ms. BlueBottle’s median at 100k is 21 366.9 ms globally and 20 850.0 ms in North America. It maintains bounded latency at 50k and 100k but falls behind the incoming rate at 250k and 500k on both testbeds, so those points are omitted. Multimmit completes all four scenarios through 1M. These loss results for BlueBottle concern the tested implementation and its application-level recovery. Its published evaluation does not include message-loss experiments. They do not establish a failure of its consensus protocol under its stated delivery assumptions.

³⁹Raptr’s published evaluation reaches 260k tx/s across 100 validators, which is roughly 2.6k tx/s originated per validator. Reaching 250k tx/s across 50 validators asks each to originate 5k tx/s. The network load per validator is essentially independent of the validator count and sits far below the hosts’ limits here, so bandwidth is not the explanation. Our best guess is the bounded pipeline of batches each validator may have in flight. With at most five batches of 450 transactions outstanding, a validator completes at most 2,250 transactions per batch-certification round trip, or roughly 4,500 tx/s when certification takes half a second. This matches the pattern we observe. The global testbed overloads at 250k, where each validator needs 5k tx/s against long round trips, while the North American testbed sustains 500k, where round trips are an order of magnitude shorter.

A single crash. A single crash changes little for Multimmit and BlueBottle. Multimmit’s global median at 500k moves from 384.8 to 391.5 ms and BlueBottle’s from 2652.0 to 2571.3 ms. Raptr’s tail is more sensitive. Globally at 100k, one crash raises the P99 from 909.3 to 1394.7 ms, with a median of 571.2 ms. In North America at 500k, its P99 rises from 367.5 to 1080.7 ms, with a median of 338.6 ms. We omit this scenario from the figures and report it in full in the released dataset.

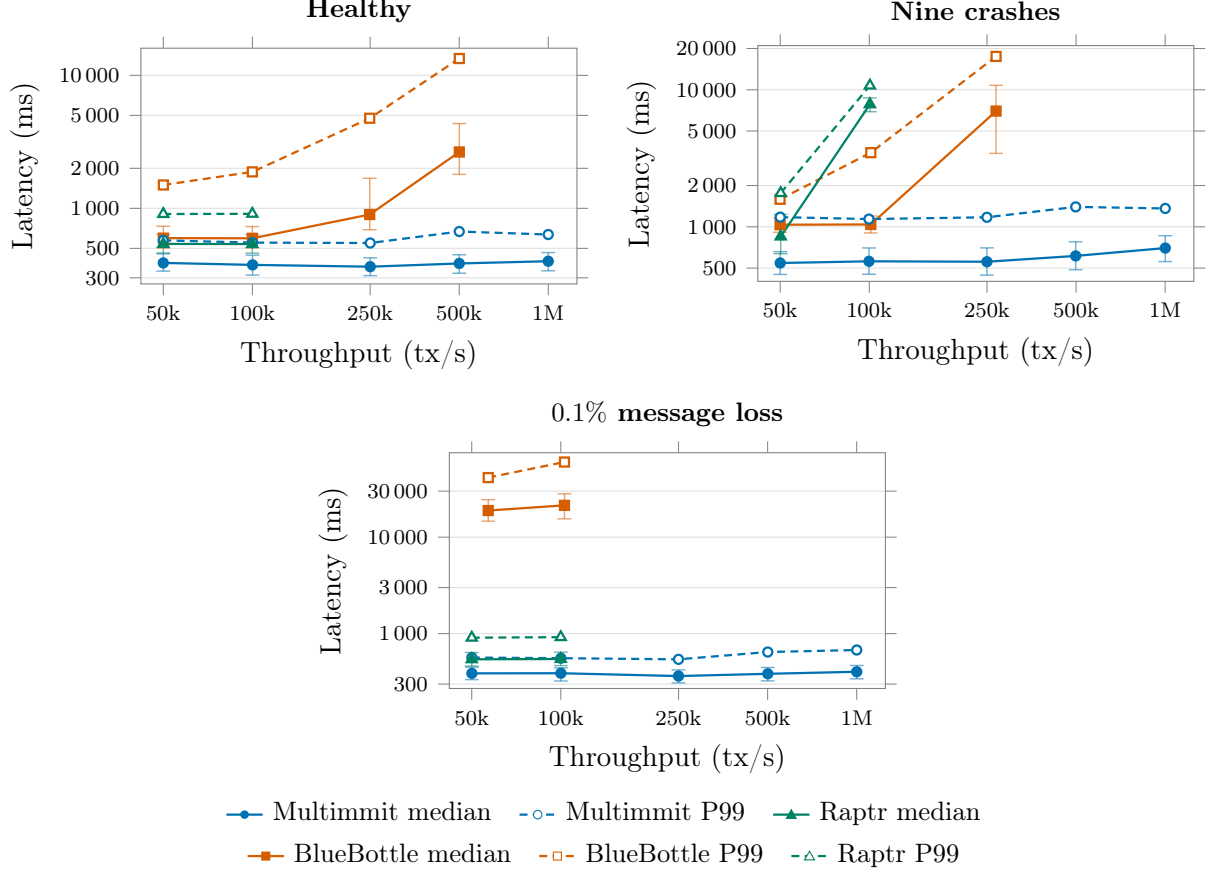


Figure 5: Global deployment with fsync disabled. Solid lines show median latency, with bars spanning P25 to P75. Dashed lines show P99. Both axes are logarithmic. Missing BlueBottle points correspond to failed runs; missing Raptr points correspond to overload exclusions and rates not offered, as described in the text.

7.3 Consensus traffic

Figure 7 shows consensus traffic in healthy operation, summed across all validators. Multimmit’s consensus counter includes headers, votes, and certificates. Repair traffic is separate and includes recovered bodies. BlueBottle counts retransmitted payloads as bodies and retrieval metadata as consensus traffic. Raptr counts serialized messages on its batch-dissemination and retrieval channels as body traffic, and messages on its compressed consensus channels as consensus traffic. Its body traffic includes the associated metadata. All three splits exclude transport overhead and are not byte-for-byte equivalent.

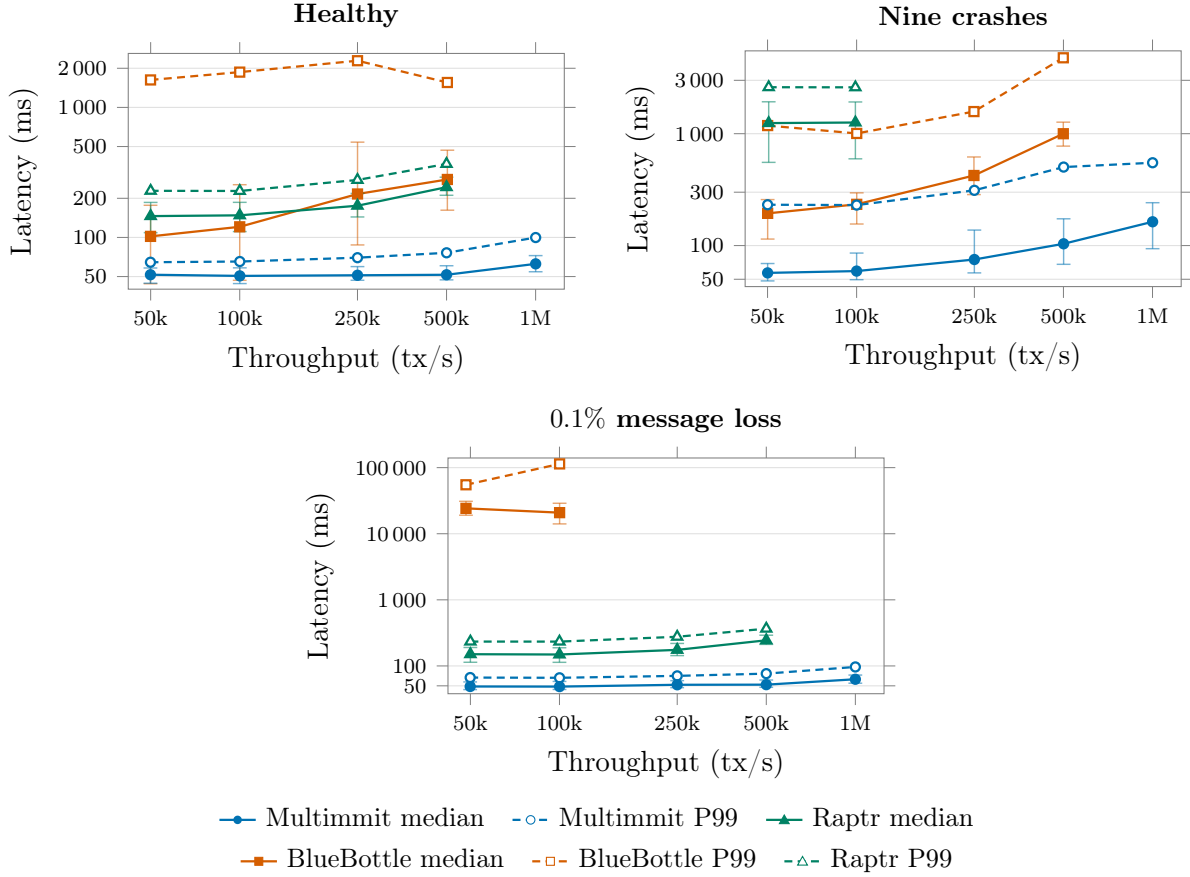


Figure 6: North American deployment with fsync disabled. Solid lines show median latency, with bars spanning P25 to P75. Dashed lines show P99. Both axes are logarithmic. Missing BlueBottle points correspond to failed runs; missing Raptr points correspond to overload exclusions and rates not offered, as described in the text.

Block bodies dominate and are nearly identical for Multimitt and BlueBottle, as expected for a shared payload. At 500k on the global testbed, Multimitt and BlueBottle transmit 12.55 and 12.56 GB/s of bodies. Their consensus rates there are 301.7 and 304.1 MB/s, which for Multimitt is 2.3% of its measured application traffic. In North America at 500k, bodies are 12.55 and 12.62 GB/s, and consensus rates are 433.6 and 403.6 MB/s, or 3.3% for Multimitt. The fault and loss scenarios change these rates only modestly for Multimitt and BlueBottle. The figure shows healthy operation and the released dataset has the rest.

In healthy global operation at 100k, Raptr transmits body traffic at 2.57 GB/s and consensus traffic at 15.2 MB/s. At 500k in North America, the corresponding rates are 12.59 GB/s and 47.3 MB/s. The lower consensus rates coexist with different quorum sizes and message formats.

Multimitt's larger certificates describe different producer prefixes, allowing progress without waiting for every producer. Their cost is small relative to body dissemination in these measurements. The counters include other consensus messages as well, so the differences between implementations cannot be attributed to certificate size alone.

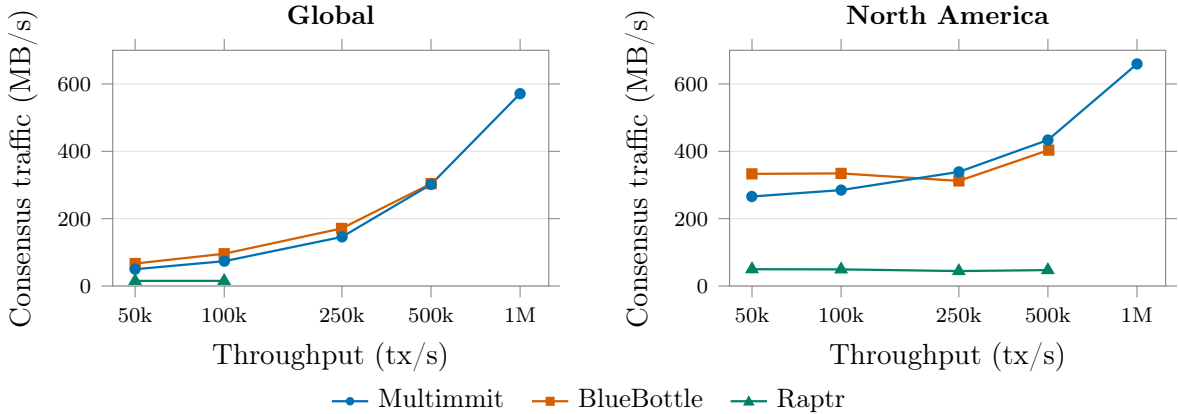


Figure 7: Consensus traffic in healthy operation, summed across all validators, with fsync disabled. Block bodies are excluded and discussed in the text.

8 Related work

Classical Byzantine consensus. The study of consensus in the presence of Byzantine faults was introduced by Lamport, Shostak and Pease [45], with a treatment of SMR given later by Schneider [62]; reliable broadcast under asynchrony is due to Bracha [14]. Dwork, Lynch and Stockmeyer [28] showed that $n \geq 3f + 1$ is optimal for partial synchrony. Standard protocols using this assumption, such as PBFT [21, 22], Tendermint [16, 17], HotStuff [82] and Simplex [23], satisfy 3-round finality, which is optimal at that resilience [4]. A large literature refines the leader-based form, including pipelining and proposal timing [27, 37, 38, 33, 74, 64], communication topology [57, 52], multi-leader deployments [73], awareness of network conditions [25], and the synchronous setting [3].

Finality in one round of voting. A long line of work [15, 43, 30, 55, 35, 69, 59] considers protocols with a ‘fast path’, allowing quick finalisation under favourable conditions. FaB [55] introduced a parameterised model with $n \geq 3f + 2p + 1$ processors, achieving finality in one round of voting when at most p processors are Byzantine; Kuznetsov, Tonkikh and Zhang [44] later showed that $n \geq 3f + 2p - 1$ is optimal (see also [2], which identifies bugs in several protocols of this family, including Zyzzyva [42]; SBFT [36] extends the approach to tolerate crashes on the fast path). Recent protocols at the $n \geq 5f + 1$ design point include Minimmit [24], whose consensus skeleton we inherit, Alpenglow [41], Kudzu [65], Hydrangea [67, 66], Banyan [77] and ChonkyBFT [29]. Flexible and multi-threshold quorums [53, 56, 79] and resilience beyond optimal bounds [50] study related trade-offs between thresholds and guarantees.

Multi-proposer dissemination. Narwhal [26] demonstrated that separating dissemination from ordering, with all processors sharing responsibility for transmitting transactions, yields far higher throughput than passing data through leaders. DAG-based protocols [39, 72, 40, 31, 11, 54, 70, 7, 68] reach consensus on the graph structure itself, with a line of work reducing their round-latency towards that of leader-based designs; Mysticeti [9] does so by referencing uncertified data, at the robustness cost measured in [7, 75] and discussed in Section 1. Fides [80] obtains censorship resistance in this setting using trusted hardware. Odontoceti [76] and BlueBottle [78] bring the $n \geq 5f + 1$ assumption to uncertified DAGs, committing in two rounds of voting.

Closest to this paper are the multi-chain designs. Autobahn [34] is our starting point: producer chains with availability certificates, cut through by leader blocks, with certification priced into every transaction’s latency. Raptr [75], built on Quorum Store [71] and Jolteon [32], is the state of the art treated at length in Sections 1 and 3. Cadence [10], concurrent work, runs a separate consensus instance per slot, with several proposers per slot disseminating proposals independently; validators vote yes or no on each proposal’s arrival, then commit-vote on a deterministic merge of the accepted proposals, in three communication rounds at $n \geq 3f + 1$, with a short-term censorship-resistance guarantee. An offline proposer costs only its own contribution, but partial dissemination or equivocation by one proposer sends the whole slot to a fallback path; Multimmit’s per-chain votes confine equivocation too, not only silence, to the offending producer’s own chain. Cadence also assumes synchronised clocks, and, its proposals being mutually hidden and merged by fee order, forgoes *predictable validity* in the sense of Gatling [61]: a proposer cannot validate transactions against the state at which they will execute.

Inclusion guarantees. AMP [20] runs dedicated proposers over Tendermint at $n \geq 3f + 1$, with validators attesting the payloads they have received in *vote extensions* attached to their consensus votes. The mechanism resembles our extension votes, but with significant differences. In AMP, the attestations cast in one height constrain only the *next* height’s proposal: a payload attested by all correct validators at height h must be included at height $h + 1$, and a payload reaching fewer correct validators may be omitted. Fresh data is thus never finalised in the height whose votes attest it, and the good-case path from payload dissemination to finalisation is five message delays. In Multimmit, extension votes finalise fresh blocks within the view whose votes endorse them, two message delays after dissemination at best, and the block inclusion theorem settles ledger membership in-view even when in-view placement fails. Abraham, Efron and Ren [1] prove a two-round latency cost inherent to inclusion guarantees of AMP’s next-block form; Multimmit’s guarantee is shaped differently, settling membership within the view, with a place in the total ordering arriving immediately in the good case, or else via the next finalised leader block. Hermes [60], concurrent work, shares Minimmit’s quorum sizes and targets the tail rather than censorship: every processor proposes at view entry, votes require comparability rather than equality, and the view of a failed leader itself finalises the heaviest common prefix of $n - f$ fallback votes, where Multimmit nullifies (though see the backup-proposal extension of Section 6). Hermes’s votes are for a single value, so the common prefix excludes the fresh suffix on which honest proposals differ, and one lagging vote truncates what a view finalises; Multimmit’s votes are per chain, confining each lag to its own chain’s coordinate. The availability machinery of Hermes covers consensus values, with payload availability inherited from the dissemination layer beneath, while Multimmit’s votes attest per-chain possession, making availability a theorem (Section 5).

Bandwidth and models. Cachin and Tessaro [19] introduced verifiable information dispersal, combining erasure codes with commitments so that data can be recovered from any sufficiently large set of verifiable fragments; see also [6, 81]. The Pipes framework [47] analyses how the data expansion rate of erasure coding governs throughput in leader-based protocols, and Carnot [48] establishes the achievable rates at 2-round and 3-round finality. Multimmit removes the leader bottleneck by different means, dissemination travelling once from each producer to all, and Section 9 discusses the regimes in which the two approaches meet. Finality gadgets and dynamic participation [18, 58, 49, 46, 51] treat settings beyond the fixed-committee model assumed here.

Our cryptographic components are standard [13, 12, 63, 5].

9 Final comments

We have presented Multimmit, established consistency, liveness and the availability, inclusion and localisation guarantees, and quantified its latency. We highlight some questions that the paper leaves open.

Limitations. Three deployment matters sit outside the analysis. First, resource bounds: as specified, `ProduceNext` places no bound on a transaction block’s size, and correct processors DA-vote, store and serve every certified block. A deployment must bound block size and admission at the mempool layer, and the analysis is indifferent to any such bound, since liveness requires only that every pending transaction is eventually included. Second, worst-case consensus data: the tens-of-kilobytes-per-view object sizes of Section 3.6 are the good case, and maximally deviant votes inflate QCs to roughly half a megabyte per view at $K = 100$ (Section 3.7), a bandwidth exposure under f Byzantine voters that implementations should provision for. Third, setup: the threshold schemes require distributed key generation, and the validator set is fixed throughout; Section 6 describes an instantiation without threshold schemes, under which a membership change reduces to an update of a key list.

Accountability. The classification of misbehaviour in Section 5 stops short of a formal treatment. Formalising the guarantee, in the style of the literature on BFT forensics, and specifying the challenge mechanism and blame rules sketched in Section 6, would put the incentive-level discussion of greedy leaders on the same footing as the protocol’s other guarantees.

Bandwidth. Multimmit removes the leader’s bandwidth as a throughput bottleneck, since transaction data travels only from each producer to all. When the number of chains is comparable to n , the load is balanced and each processor receives each byte of transaction data once, so there is no analogue of the data expansion that erasure coding exists to remove in the leader-based setting [48]. When the number of chains is small, each producer’s outgoing bandwidth bounds the rate at which its own chain can grow, and the trade-offs begin to resemble those of the leader-based setting; we leave the comparison of the two regimes, along with the further optimisations listed in Section 6, for future work. Finally, we have assumed for simplicity that the block producers are the validators, and the intended deployments separate the two roles; the generalisation, with K producer chains for K unrelated to n , changes nothing of substance in the analysis.

A The cost of durability

A coordinated upgrade or a common fault can stop every validator. Replication in memory alone does not protect against this event, so implementations must persist both their signing decisions and the data whose availability they promise. We measure Multimmit with `fsync` enabled to quantify this cost. Recovery assumes that durable storage survives the shutdown. These measurements do not themselves test a full-cluster restart.

Multimmit waits for the block body and custody metadata to become durable before exposing a producer header or issuing a data-availability share. It also waits for the safety journal to

persist each signing choice before exposing the resulting signature. This covers producer headers, availability shares, leader proposals, votes, and the novote and nullify decisions used on timeout. Adjacent journal records share a sync. Checkpoints become durable before older journal records are pruned, and finalised archives and delivery acknowledgements are synced before their durable progress is published. Forwarding an already verifiable certificate does not require a new fsync.

Figure 8 compares the two durability modes in healthy operation, under the setup of Section 7. Globally at 1M, enabling fsync moves the median from 400.0 to 451.9ms and the P99 from 635.2 to 800.5ms. In North America at 1M, it moves the median from 62.6 to 80.6ms and the P99 from 99.8 to 128.3ms. All cohorts complete with fsync enabled, in every scenario. The cost under faults is similar. The largest tail effect is at 1M with nine crashes globally, where the P99 is 2661.2ms with fsync enabled against 1361.3ms without. Batching lets multiple bodies and journal records share durability barriers. The observed differences also include variation between runs. Full statistics for both durability modes appear in the released dataset.

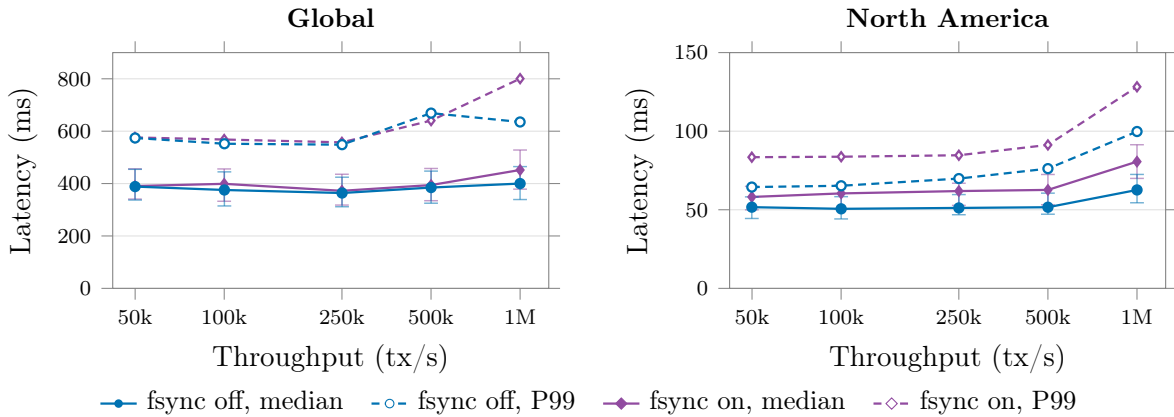


Figure 8: The cost of durability in healthy operation. Multimitt with fsync enabled against fsync disabled. Solid lines show median latency, with bars spanning P25 to P75. Dashed lines show P99. Vertical axes are linear and start at zero.

References

- [1] Ittai Abraham, Yuval Efron, and Ling Ren. The latency cost of censorship resistance. Cryptology ePrint Archive, Paper 2025/2136, 2025.
- [2] Ittai Abraham, Guy Gueta, Dahlia Malkhi, Lorenzo Alvisi, Rama Kotla, and Jean-Philippe Martin. Revisiting fast practical byzantine fault tolerance. *arXiv preprint arXiv:1712.01367*, 2017.
- [3] Ittai Abraham, Dahlia Malkhi, Kartik Nayak, Ling Ren, and Maofan Yin. Sync hotstuff: Simple and practical synchronous state machine replication. In *2020 IEEE Symposium on Security and Privacy (SP)*, pages 106–118. IEEE, 2020.
- [4] Ittai Abraham, Kartik Nayak, Ling Ren, and Zhuolun Xiang. Good-case latency of byzantine broadcast: A complete categorization. In *Proceedings of the 2021 ACM Symposium on Principles of Distributed Computing*, pages 331–341, 2021.

- [5] Mark Abspoel, Thomas Attema, and Matthieu Rambaud. Malicious security comes for free in consensus with leaders. *Cryptology ePrint Archive*, 2020.
- [6] Nicolas Alhaddad, Sisi Duan, Mayank Varia, and Haibin Zhang. Succinct erasure coding proof systems. *Cryptology ePrint Archive*, 2021.
- [7] Balaji Arun, Zekun Li, Florian Suri-Payer, Sourav Das, and Alexander Spiegelman. Shoal++: High throughput DAG BFT can be fast! *arXiv preprint arXiv:2405.20488*, 2024.
- [8] Pierre-Louis Aublin, Sonia Ben Mokhtar, and Vivien Quéma. RBFT: Redundant Byzantine fault tolerance. In *2013 IEEE 33rd International Conference on Distributed Computing Systems*, pages 297–306. IEEE, 2013.
- [9] Kushal Babel, Andrey Chursin, George Danezis, Anastasios Kichidis, Lefteris Kokoris-Kogias, Arun Koshy, Alberto Sonnino, and Mingwei Tian. Mysticeti: Reaching the limits of latency with uncertified dags. *arXiv preprint arXiv:2310.14821*, 2023.
- [10] Kushal Babel, Fatima Elsheimy, Lioba Heimbach, Mohammad Mussadiq Jalalzai, Tobias Klenze, Jovan Komatovic, Jason Milionis, Mike Setrin, and Victor Shoup. Cadence: Extreme pipelining with multiple concurrent proposers. *arXiv preprint arXiv:2607.02275*, 2026.
- [11] Leemon Baird. The Swirls hashgraph consensus algorithm: Fair, fast, Byzantine fault tolerance. Technical report, Swirls Tech Report SWIRLDS-TR-2016-01, 2016.
- [12] Dan Boneh, Craig Gentry, Ben Lynn, and Hovav Shacham. Aggregate and verifiably encrypted signatures from bilinear maps. In *Advances in Cryptology—EUROCRYPT 2003*. Springer, 2003.
- [13] Dan Boneh, Ben Lynn, and Hovav Shacham. Short signatures from the weil pairing. In *International conference on the theory and application of cryptology and information security*, pages 514–532. Springer, 2001.
- [14] Gabriel Bracha. Asynchronous Byzantine agreement protocols. *Information and Computation*, 75(2):130–143, 1987.
- [15] Francisco Brasileiro, Fabíola Greve, Achour Mostéfaoui, and Michel Raynal. Consensus in one communication step. In *International Conference on Parallel Computing Technologies*, pages 42–50. Springer, 2001.
- [16] Ethan Buchman. *Tendermint: Byzantine fault tolerance in the age of blockchains*. PhD thesis, University of Guelph, 2016.
- [17] Ethan Buchman, Jae Kwon, and Zarko Milosevic. The latest gossip on bft consensus. *arXiv preprint arXiv:1807.04938*, 2018.
- [18] Vitalik Buterin and Virgil Griffith. Casper the friendly finality gadget. *arXiv preprint arXiv:1710.09437*, 2017.
- [19] Christian Cachin and Stefano Tessaro. Asynchronous verifiable information dispersal. In *24th IEEE Symposium on Reliable Distributed Systems (SRDS’05)*, pages 191–201. IEEE, 2005.

- [20] Daniel Cason, Gordon Liao, Sergio Mena, Nenad Milošević, Adi Seredinschi, Alessandro Sforzin, João Sousa, and Preston Vander Vos. AMP: Arc multi-proposer protocol with bounded inclusion guarantees. *arXiv preprint arXiv:2605.23677*, 2026.
- [21] Miguel Castro and Barbara Liskov. Practical byzantine fault tolerance. In *OSDI*, volume 99, pages 173–186, 1999.
- [22] Miguel Castro and Barbara Liskov. Practical byzantine fault tolerance and proactive recovery. *ACM Transactions on Computer Systems (TOCS)*, 20(4):398–461, 2002.
- [23] Benjamin Y Chan and Rafael Pass. Simplex consensus: A simple and fast consensus protocol. In *Theory of Cryptography Conference*, pages 452–479. Springer, 2023.
- [24] Brendan Kobayashi Chou, Andrew Lewis-Pye, and Patrick O’Grady. Minimit: Fast finality with even faster blocks. *arXiv preprint arXiv:2508.10862*, 2025.
- [25] Shir Cohen, Rati Gelashvili, Lefteris Kokoris-Kogias, Zekun Li, Dahlia Malkhi, Alberto Sonnino, and Alexander Spiegelman. Be aware of your leaders. In *International Conference on Financial Cryptography and Data Security*. Springer, 2022.
- [26] George Danezis, Lefteris Kokoris-Kogias, Alberto Sonnino, and Alexander Spiegelman. Narwhal and tusk: a dag-based mempool and efficient bft consensus. In *Proceedings of the Seventeenth European Conference on Computer Systems*, pages 34–50, 2022.
- [27] Isaac Doidge, Raghavendra Ramesh, Nibesh Shrestha, and Joshua Tobkin. Moonshot: Optimizing chain-based rotating leader bft via optimistic proposals. *arXiv preprint arXiv:2401.01791*, 2024.
- [28] Cynthia Dwork, Nancy A. Lynch, and Larry Stockmeyer. Consensus in the presence of partial synchrony. *Journal of the ACM*, 35(2):288–323, 1988.
- [29] Bruno França, Denis Kolegov, Igor Konnov, and Grzegorz Prusak. Chonkybft: Consensus protocol of zksync. *arXiv preprint arXiv:2503.15380*, 2025.
- [30] Roy Friedman, Achour Mostefaoui, and Michel Raynal. Simple and efficient oracle-based consensus protocols for asynchronous byzantine systems. *IEEE Transactions on Dependable and Secure Computing*, 2(1):46–56, 2005.
- [31] Adam Gagol and Michał Świątek. Aleph: A leaderless, asynchronous, byzantine fault tolerant consensus protocol. *arXiv preprint arXiv:1810.05256*, 2018.
- [32] Rati Gelashvili, Lefteris Kokoris-Kogias, Alberto Sonnino, Alexander Spiegelman, and Zhuolun Xiang. Jolteon and Ditto: Network-adaptive efficient consensus with asynchronous fallback. In *International Conference on Financial Cryptography and Data Security*. Springer, 2022.
- [33] Neil Giridharan, Heidi Howard, Ittai Abraham, Natacha Crooks, and Alin Tomescu. No-commit proofs: Defeating livelock in BFT. *Cryptology ePrint Archive*, 2021.
- [34] Neil Giridharan, Florian Suri-Payer, Ittai Abraham, Lorenzo Alvisi, and Natacha Crooks. Autobahn: Seamless high speed bft. In *Proceedings of the ACM SIGOPS 30th Symposium on Operating Systems Principles*, pages 1–23, 2024.

- [35] Rachid Guerraoui and Marko Vukolić. Refined quorum systems. In *Proceedings of the twenty-sixth annual ACM symposium on Principles of distributed computing*, pages 119–128, 2007.
- [36] Guy Golan Gueta, Ittai Abraham, Shelly Grossman, Dahlia Malkhi, Benny Pinkas, Michael Reiter, Dragos-Adrian Seredinschi, Orr Tamir, and Alin Tomescu. Sbft: A scalable and decentralized trust infrastructure. In *2019 49th Annual IEEE/IFIP international conference on dependable systems and networks (DSN)*, pages 568–580. IEEE, 2019.
- [37] Mohammad M Jalalzai, Jianyu Niu, Chen Feng, and Fangyu Gai. Fast-HotStuff: A fast and robust BFT protocol for blockchains. *IEEE Transactions on Dependable and Secure Computing*, 2023.
- [38] Dakai Kang, Suyash Gupta, Dahlia Malkhi, and Mohammad Sadoghi. HotStuff-1: Linear consensus with one-phase speculation. *arXiv preprint arXiv:2408.04728*, 2024.
- [39] Idit Keidar, Eleftherios Kokoris-Kogias, Oded Naor, and Alexander Spiegelman. All you need is dag. In *Proceedings of the 2021 ACM Symposium on Principles of Distributed Computing*, pages 165–175, 2021.
- [40] Idit Keidar, Oded Naor, Ouri Poupko, and Ehud Shapiro. Cordial miners: Fast and efficient consensus for every eventuality. *arXiv preprint arXiv:2205.09174*, 2022.
- [41] Quentin Kniep, Jakub Sliwinski, and Roger Wattenhofer. Solana alpenglw consensus. <https://www.scribd.com/document/895233790/Solana-Alpenglow-White-Paper>, 2025.
- [42] Ramakrishna Kotla, Lorenzo Alvisi, Mike Dahlin, Allen Clement, and Edmund Wong. Zyzzyva: speculative byzantine fault tolerance. In *Proceedings of twenty-first ACM SIGOPS symposium on Operating systems principles*, pages 45–58, 2007.
- [43] Klaus Kursawe. Optimistic byzantine agreement. In *21st IEEE Symposium on Reliable Distributed Systems, 2002. Proceedings.*, pages 262–267. IEEE, 2002.
- [44] Petr Kuznetsov, Andrei Tonkikh, and Yan X Zhang. Revisiting optimal resilience of fast byzantine consensus. In *Proceedings of the 2021 ACM Symposium on Principles of Distributed Computing*, pages 343–353, 2021.
- [45] Leslie Lamport, Robert Shostak, and Marshall Pease. The byzantine generals problem. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 4(3):382–401, 1982.
- [46] Andrew Lewis-Pye, Dahlia Malkhi, Oded Naor, and Kartik Nayak. Lumiere: Making optimal bft for partial synchrony practical. In *Proceedings of the 43rd ACM Symposium on Principles of Distributed Computing*, pages 135–144, 2024.
- [47] Andrew Lewis-Pye, Kartik Nayak, and Nibesh Shrestha. The pipes model for latency analysis. *Cryptology ePrint Archive*, 2025.
- [48] Andrew Lewis-Pye and Patrick O’Grady. The Carnot bound: Limits and possibilities for bandwidth-efficient consensus. *arXiv preprint arXiv:2603.11797*, 2026.

- [49] Andrew Lewis-Pye and Tim Roughgarden. Permissionless consensus. *arXiv preprint arXiv:2304.14701*, 2023.
- [50] Andrew Lewis-Pye and Tim Roughgarden. Beyond optimal fault tolerance. *arXiv preprint arXiv:2501.06044*, 2025.
- [51] Andrew Lewis-Pye and Ehud Shapiro. Morpheus consensus: Excelling on trails and autobahns. *arXiv preprint arXiv:2502.08465*, 2025.
- [52] Marta Likhava, Giuliano Losa, David Mazières, Graydon Hoare, Nicolas Barry, Eli Gafni, Jonathan Jove, Rafał Malinowsky, and Jed McCaleb. Fast and secure global payments with Stellar. In *Proceedings of the 27th ACM Symposium on Operating Systems Principles*, 2019.
- [53] Dahlia Malkhi, Kartik Nayak, and Ling Ren. Flexible Byzantine fault tolerance. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, 2019.
- [54] Dahlia Malkhi, Chrysoula Stathakopoulou, and Maofan Yin. BBKA-chain: One-message, low latency BFT consensus on a DAG. *arXiv preprint arXiv:2310.06335*, 2023.
- [55] J-P Martin and Lorenzo Alvisi. Fast byzantine consensus. *IEEE Transactions on Dependable and Secure Computing*, 3(3):202–215, 2006.
- [56] Atsuki Momose and Ling Ren. Multi-threshold Byzantine fault tolerance. In *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security*, 2021.
- [57] Ray Neiheiser, Miguel Matos, and Luís Rodrigues. Kauri: Scalable BFT consensus with pipelined tree-based dissemination and aggregation. In *Proceedings of the ACM SIGOPS 28th Symposium on Operating Systems Principles*, 2021.
- [58] Joachim Neu, Ertem Nusret Tas, and David Tse. Ebb-and-flow protocols: A resolution of the availability-finality dilemma. In *2021 IEEE Symposium on Security and Privacy (SP)*, pages 446–465. IEEE, 2021.
- [59] Rafael Pass and Elaine Shi. Thunderella: Blockchains with optimistic instant confirmation. In *Advances in Cryptology–EUROCRYPT 2018: 37th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Tel Aviv, Israel, April 29-May 3, 2018 Proceedings, Part II 37*, pages 3–33. Springer, 2018.
- [60] Alejandro Ranchal-Pedrosa, Dakai Kang, Neil Giridharan, Mohammad Sadoghi, Dahlia Malkhi, and Ben Marsh. Hermes: Low tail-latency via prefix consensus. *arXiv preprint arXiv:2607.25916*, 2026.
- [61] Giulia Scaffino, Max Resnick, and Joachim Neu. Gatling: Rapid-fire consensus from parallel composition. *arXiv preprint arXiv:2606.18220*, 2026.
- [62] Fred B Schneider. Implementing fault-tolerant services using the state machine approach: A tutorial. *ACM Computing Surveys (CSUR)*, 22(4):299–319, 1990.
- [63] Victor Shoup. Practical threshold signatures. In *International conference on the theory and applications of cryptographic techniques*, pages 207–220. Springer, 2000.

- [64] Victor Shoup. Sing a song of simplex. *Cryptology ePrint Archive*, 2023.
- [65] Victor Shoup, Jakub Sliwinski, and Yann Vonlanthen. Kudzu: Fast and simple high-throughput bft. *arXiv preprint arXiv:2505.08771*, 2025.
- [66] Nibesh Shrestha and Aniket Kate. Hydrangea++: Enhancing hydrangea with optimistic proposals. White paper, <https://supra.com/documents/hydrangea-plus-plus.pdf>, 2025.
- [67] Nibesh Shrestha, Aniket Kate, and Kartik Nayak. Hydrangea: Optimistic two-round partial synchrony with improved fault resilience. *Cryptology ePrint Archive*, 2025.
- [68] Nibesh Shrestha, Rohan Shrothrium, Aniket Kate, and Kartik Nayak. Sailfish: Towards improving the latency of dag-based bft. In *2025 IEEE Symposium on Security and Privacy (SP)*, pages 1928–1946. IEEE, 2025.
- [69] Yee Jiun Song and Robbert Van Renesse. Bosco: One-step byzantine asynchronous consensus. In *International Symposium on Distributed Computing*, pages 438–450. Springer, 2008.
- [70] Alexander Spiegelman, Balaji Arun, Rati Gelashvili, and Zekun Li. Shoal: Improving dag-bft latency and robustness. In *International Conference on Financial Cryptography and Data Security*, pages 92–109. Springer, 2024.
- [71] Alexander Spiegelman and Brian Cho. Quorum Store: How consensus horizontally scales on the Aptos blockchain. Aptos Labs blog, <https://medium.com/aptoslabs>, 2023.
- [72] Alexander Spiegelman, Neil Giridharan, Alberto Sonnino, and Lefteris Kokoris-Kogias. Bullshark: Dag bft protocols made practical. In *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security*, pages 2705–2718, 2022.
- [73] Chrysoula Stathakopoulou, Tudor David, and Marko Vukolić. Mir-BFT: High-throughput BFT for blockchains. *arXiv preprint arXiv:1906.05552*, 2019.
- [74] Xiao Sui, Sisi Duan, and Haibin Zhang. Marlin: Two-phase BFT with linearity. In *2022 52nd Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*. IEEE, 2022.
- [75] Andrei Tonkikh, Balaji Arun, Zhuolun Xiang, Zekun Li, and Alexander Spiegelman. Raptr: Prefix consensus for robust high-performance bft. *arXiv preprint arXiv:2504.18649*, 2025.
- [76] Preston Vander Vos. Odontoceti: Ultra-fast DAG consensus with two round commitment. *arXiv preprint arXiv:2510.01216*, 2025.
- [77] Yann Vonlanthen, Jakub Sliwinski, Massimo Albarello, and Roger Wattenhofer. Banyan: Fast rotating leader bft. In *Proceedings of the 25th International Middleware Conference*, pages 494–507, 2024.
- [78] Preston Vander Vos, Alberto Sonnino, Giorgos Tsimos, Philipp Jovanovic, and Lefteris Kokoris-Kogias. Bluebottle: Fast and robust blockchains through subsystem specialization. *arXiv preprint arXiv:2511.15361*, 2025.

- [79] Zhuolun Xiang, Dahlia Malkhi, Kartik Nayak, and Ling Ren. Strengthened fault tolerance in Byzantine fault tolerant replication. In *2021 IEEE 41st International Conference on Distributed Computing Systems (ICDCS)*. IEEE, 2021.
- [80] Shaokang Xie, Dakai Kang, Hanzheng Lyu, Jianyu Niu, and Mohammad Sadoghi. Fides: Scalable censorship-resistant DAG consensus via trusted components. *arXiv preprint arXiv:2501.01062*, 2025.
- [81] Lei Yang, Seo Jin Park, Mohammad Alizadeh, Sreeram Kannan, and David Tse. DispersedLedger: High-throughput Byzantine consensus on variable bandwidth networks. In *19th USENIX Symposium on Networked Systems Design and Implementation (NSDI 22)*, 2022.
- [82] Maofan Yin, Dahlia Malkhi, Michael K Reiter, Guy Golan Gueta, and Ittai Abraham. Hotstuff: Bft consensus with linearity and responsiveness. In *Proceedings of the 2019 ACM Symposium on Principles of Distributed Computing*, pages 347–356, 2019.