The Pipes Model for Latency and Throughput Analysis

Anonymous Author(s)

***Abstract*—Traditionally, latency in distributed computing protocols is expressed as the number of communication rounds or network delays; it does not take into account the amount of data sent or the dependencies among parties sending the data. Moreover, throughput for a protocol is typically only empirically computed. Due to this, the only means of obtaining or comparing the practical latency and throughput of protocols is through expensive implementation and experimentation.**

In this paper, we present Pipes, a model for analyzing latency and throughput in state machine replication (SMR) protocols. The Pipes model captures the effect of processor bandwidth S, transaction arrival rate D, and the network delay Δ, enabling us to explicitly specify the throughput bottleneck and the latency of a protocol. Using Pipes, we perform an analysis of broadcast primitives such as Best-effort Broadcast and Reliable Broadcast, as well as state-of-the-art SMR protocols such as DispersedSimplex, Tendermint, HotStuff, and Sailfish. We experimentally validate these results by implementing the Best-effort Broadcast primitives and SMR protocols (DispersedSimplex and Sailfish).

Our comparisons show clear trade-offs: single-sender protocols that exploit pipelining and erasure coding (e.g., DispersedSimplex) can achieve substantially lower latency across many regimes but have a lower latency bottleneck by a constant factor; many DAG-based protocols push the bottleneck higher at the cost of higher per-block latency scaling. HotStuff's leader-relay design, while communication-efficient, yields higher latency than Tendermint in our model due to leader bandwidth bottlenecks.

1. Introduction

The common timing assumptions considered in Distributed Computing are the synchronous, asynchronous, and partially synchronous settings. In the synchronous setting, messages are delivered within a known bound Δ. In the asynchronous setting [1], messages are eventually delivered but with no time bound. In the partially synchronous setting [2], there is an unknown global-stabilization-time (GST) after which messages are always delivered within the known bound Δ.

Common to all these models is the fact that message sizes are unbounded: for example, in the synchronous setting, a message of *any size* sent at time t is guaranteed to be delivered by $t + \Delta$. This assumption simplifies analysis and may not be critical for establishing consistency and liveness. However, a consequence of their indifference to message sizes is that these models give no meaningful way to analyze some other important properties of a blockchain protocol, such as the maximum throughput it can handle.

These models also give limited ability to analyze (real-world) *latency*. In the case of protocols for State-Machine-Replication (SMR) [3], this can be informally defined as the time between the first point at which a transaction is received by a correct (non-faulty) processor and the first time at which that transaction is *finalized* by all correct processors. In the standard models, 'latency' is upper-bounded simply by counting the number of rounds of communication required: if 10 rounds of communication are required in the synchronous model, then 'latency' is 10Δ. Since there is a clear understanding that this is unrealistic (sending larger amounts of data will generally take longer than sending a smaller amount), we are then committed to an awkward dance in which we also consider *communication complexity*, e.g., the number of bits of data that must be sent to achieve finalization in the case of SMR protocols, with no formal way to weigh these metrics. If a protocol has lower 'latency' (round complexity) but higher communication complexity than another, what does this mean in terms of real-world latency? Generally (but not always), it is the real-world latency that one actually cares about, rather than the round complexity or communication complexity: while potentially of interest in their own right, the latter metrics are generally used as proxies for the former.

Recent research on multi-sender protocols. These considerations are brought to a head by recent research that looks to develop SMR protocols that can handle high throughputs. Typical protocols, such as PBFT [4], Tendermint [5], [6], HotStuff [7], [8] (henceforth called 'single-sender' protocols), have a single designated (potentially rotating) 'leader' who is responsible for making and disseminating proposals to others. In a context where processors do have limited upload/download speeds, there is an estimation by many in the community that the leader should act as a bottleneck, limiting the ability to deal with high throughput. A significant amount of research has therefore investigated the use of 'multi-sender' protocols in which many processors distribute proposals in parallel. Examples include DAG-based protocols such as DAG-Rider [9], Bullshark [10], Shoal [11], Sailfish [12], Cordial Miners [13], Mysticeti [14], and variants such as Autobahn [15]. Some of these protocols serve as the consensus mechanism for major blockchains with market caps in the billions of dollars.[1] However, the models presently used in distributed computing have no way of formally analyzing the claim that their better use of

1. For example, see https://coinmarketcap.com/currencies/sui/.

network bandwidth should lead to improved performance (i.e., lower real-world latency for various throughputs, or greater achievable throughputs) when compared with single-sender protocols.

The goals of this paper. The first goal of this paper is to describe a basic model that allows one to calculate latency as a function of network bandwidth, and thereby to compare the performance of protocols using the single-sender and multi-sender approaches (as an example application). According to this model, formally described in Section 2, each processor has a bandwidth (upload/download capacity) of S (constant size) 'data parcels' per timeslot, while the network receives incoming transactions at a rate of D data parcels per timeslot. For any given protocol, one can then calculate latency as a function of S, D, network delays, the number of processors n, and some other system parameters.[2] It is also easily observed that each protocol has a *latency bottleneck*: this is an incoming transaction rate at which latency becomes unbounded over the protocol execution, i.e., a maximum throughput that the protocol can handle without unbounded latency.

With the model in place, our second goal is then to understand how latency varies with system parameters such as the desired throughput D and the number of processors n. A significant part of this analysis is to understand how the latency bottleneck (i.e., maximum possible throughput) depends on these parameters, but we are also interested in how latency varies below the bottleneck. In particular, we aim to answer the following question:

> *If each processor has a bandwidth of S parcels per time slot, if the incoming data rate is D parcels per time slot, and if there is a delay of Δ time slots between processors, can we analytically compute the trade-off between throughput and latency for a given protocol and a given number of processors?*

Simplifications made by the analysis in this paper. We focus on the 'good case' where message delivery is reliable and processors act correctly. There are at least two reasons for this focus:

1) Real-world experience (e.g., [18]) shows that the good case is actually the common case, i.e., in real-world applications, message delivery is reliable more often than not, and processors normally behave correctly. The intention of developers, therefore, is generally to build SMR protocols that remain consistent during periods of asynchrony and under substantial adversarial action, but which are optimized to give low latency in the good case, since this is the case that applies most of the time.
2) Unsurprisingly, using the new model makes latency analysis more complex. By focusing on the good-case scenario for latency, we can highlight the results that researchers and practitioners are likely most interested in, while keeping our analysis reasonably simple.

2. Some papers (e.g., [16], [17]) model network capacity, but these are tailored to gossip-based protocols and don't enable granular comparison of single vs. multi-sender approaches.

Our model is also designed with the aim of being as straightforward as possible, while still effectively capturing key factors involved in comparing latency for different protocol design approaches. To this end, we certainly over-simplify some significant considerations that will impact latency in practice. In particular, we make the following simplifying assumptions and leave more realistic modeling for future work:

- We ignore computation costs such as signature generation, verification, and erasure coding.
- We suppose upload and download speeds are equal and constant.
- We suppose all processors have the same bandwidth S.
- We suppose message delay is the same between all pairs of processors.

These simplifications mean that the model captures an idealised scenario: it establishes a lower bound on the latency that a protocol can achieve assuming (amongst other things) zero computational cost. Although we expect that future models may benefit from further elaborating on these considerations, we believe it is beneficial to start with a model that is as simple as possible, while effectively addressing critical elements necessary for evaluating the latency/throughput trade-off.

The benefits of a model. By introducing this model, we aim to take a step towards the development of a formal theory for latency analysis, the establishment of lower bounds, and the ability to predict and analyse a protocol's latency/throughput trade-off. This is particularly important in distributed computing, where:

(i) Experiments involving many processors are costly;
(ii) Results are often hard to compare and interpret due to implementation differences (models help abstract away these variations).

A high-level overview of results. With the aim of building to an analysis for state-of-the-art SMR protocols, we begin by considering protocols for simpler primitives: we consider Best-effort Broadcast (BeB), Consistent Broadcast (CB), Reliable Broadcast (RB), and RB with erasure coding. Results for these protocols are presented in Sections 3 and 4. In the interest of conserving space, we focus here on results for state-of-the-art SMR protocols.

SMR protocols. With respect to DAG-based protocols, we focus on the case of 'certified' protocols such as DAG-Rider, Bullshark, Shoal, and Sailfish, which use some form of CB/RB as the underlying method of block propagation.[3] Once the simpler primitives are analyzed, extending the latency analysis to certified DAG-based protocols is straightforward: the problem reduces to counting the number of rounds of CB or RB (with or without erasure coding) required to finalize each block.

3. The term 'certified' [19] stems from the fact that such protocols start by producing a 'certificate' for each block, prior to reaching consensus on total ordering. We leave it to future work to analyze protocols, such as Cordial Miners and Mysticeti, in which block producers just send blocks directly to all others (BeB).

To make things concrete, we focus on the example of Sailfish [12]: the analysis we carry out is also easily adapted to deal with other well-known DAG-based protocols. As far as we are aware, when CB is used as the underlying method of block propagation, Sailfish has latency that is at least on par with other existing DAG-based protocols in the 'good case'.[4] If block metadata is of size $n\lambda$ and 'votes' are of size λ, latency for Sailfish is:[5]

$$\left(\frac{9\lambda n^2}{S} + 9\Delta\right)\left(1 + \frac{8}{9((S/D)-1)}\right).$$

To understand this expression, note that, for realistic values of S, $\frac{9\lambda n^2}{S}$ will be small unless n is large, meaning that the left term in parentheses will be dominated by 9Δ. If S/D is large, then the right term in parenthesis will be roughly 1. However, as S/D approaches 1, the right term (and latency) will tend to infinity, meaning that a latency bottleneck occurs at $S = D$.

Among protocols we are aware of in the single-sender setting that tolerate $f < n/3$ faults, that with the most competitive latency is DispersedSimplex [20]. With a stable leader, if block metadata is of size M, and if votes and hashes are of size λ, the latency of DispersedSimplex is:

$$\left(\frac{6M+2n\lambda(\log(n)+4)}{S}\right)\left(1 + \frac{1}{(S/3D)-1}\right) + \frac{2n\lambda}{S} + 3\Delta.$$

To understand this expression, note that, when $S/3D$ is large, the term $\left(1 + \frac{1}{(S/3D)-1}\right)$ will be close to 1, but this term will tend to infinity as $S/3D$ approaches 1.

To illustrate the difference between these results, suppose processors can upload/download at a rate of 1 Gbps. Suppose transactions are 2500 bits (about 300 bytes, similar to typical Bitcoin transactions). In the single-sender setting, suppose M is 1000 bits. Set $\lambda = 500$ bits and $\Delta = 0.2$ seconds. The first graph in Figure 1 supposes the incoming transaction rate D is 10^5 transactions per second and shows the resulting latency (in seconds) as a function of n. The second graph (on the right) in Figure 1 fixes $n = 400$ and shows the resulting latency as a function of the number of incoming transactions per second divided by 10^4. One can see that DispersedSimplex has significantly lower latency until one reaches its latency bottleneck, but that the latency bottleneck for DispersedSimplex is a third of that for Sailfish (when Sailfish uses Consistent Broadcast as the underlying primitive for block propagation).

Tendermint and HotStuff. Unlike DispersedSimplex, most single- or multi-sender protocols do not use erasure coding. It is therefore useful to also compare standard protocols such as Tendermint and HotStuff with Sailfish. We analyze latency for Tendermint and Hotstuff in Appendices L and M. Figure 2 illustrates the corresponding results, for the same setup assumptions as described above. The first graph in Figure 2 supposes the incoming transaction rate D is 2000 transactions per second and shows the resulting latency (in seconds) as a function of n. The second graph (on the right) fixes $n = 30$ and shows the resulting latency as a function of the number of incoming transactions per second divided by 10^3. One can see that the reduced communication complexity of HotStuff when compared to Tendermint does not result in lower latency: in fact, latency for HotStuff is strictly greater than latency for Tendermint for all parameter values.

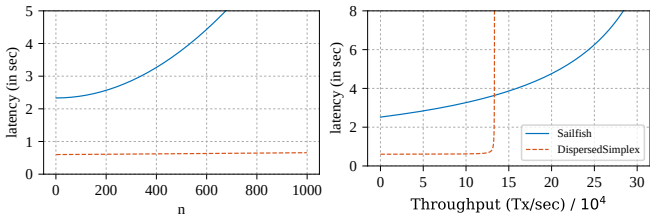

Figure 1: Latency for Sailfish and DispersedSimplex: parameters are explained in Section 1.

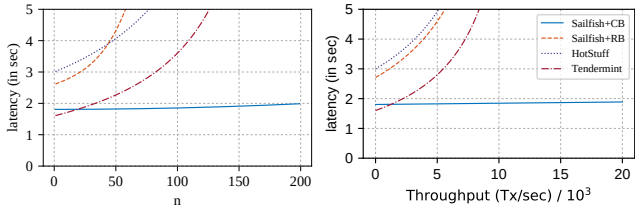

Figure 2: Comparing latency for DAG-based and standard SMR protocols: parameters are explained in Section 1.

Summary. For state-of-the-art SMR protocols, the picture that emerges is one with trade-offs. If one compares single-sender protocols that use pipelining and erasure coding, such as DispersedSimplex, with DAG-based protocols such as Sailfish or Bullshark, the former are seen to have lower latency for a wide range of throughputs, while the benefit of the latter protocols is that they have a latency bottleneck which is higher by a constant factor.

Paper structure. Section 2 defines the model. Section 3 considers Best-effort Broadcast. Section 4 considers CB, RB, and RB with erasure coding. Section 5 considers SMR protocols. Section 6 describes the results of experiments to test the model. Section 7 describes related work and Section 8 has some final comments.

2. Model

The system of processors. We consider a system of n processors $\{p_1, \ldots, p_n\}$, where each processor maintains a direct connection with every other processor. Processors communicate with each other using reliable delivery-in-order channels. In practice, TCP provides such a functionality. We assume all communication happens in the form of information parcels, *each of which is some fixed (constant)*

4. A recent paper [19], published as we were finalizing this draft, proposes Starfish, which may achieve lower latency than Sailfish in certain settings. We leave a detailed analysis of Starfish to future work, but note that, qualitatively, its analysis should resemble that of Sailfish when Reliable Broadcast with erasure-coded block propagation is used.

5. The 9Δ term in the left parentheses results because we use a stricter latency definition than some prior work. See Section 5.1 for details.

number of bits.[6] Processors may be *correct*, meaning that they execute protocol instructions as prescribed, or may display various faults (e.g., be *Byzantine*), but we will focus on the performance of protocols when processors are correct in what follows.

Cryptographic assumptions. We assume standard cryptographic primitives (signatures, PKI, threshold signatures, erasure coding, a hash function H) and a computationally bounded adversary. Following a common standard in distributed computing and for simplicity of presentation (to avoid the analysis of certain 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 these cryptographic schemes.

Message delays. For simplicity, divide time into discrete slots and assume a uniform message delay of Δ slots between any two processors.[7] While real systems may operate under partial synchrony or asynchrony [2], our goal is to analyze throughput and latency under optimistic conditions, making the synchronous setting a reasonable assumption.

The upload and download buffers. Intuitively, each processor has a 'bandwidth' of S parcels per time slot, i.e., it is capable of uploading S parcels and downloading S parcels in each time slot. To formalize this, we consider each processor to have a single upload buffer and a single download buffer, from which it can upload and download parcels of data. When a processor executes an instruction to send a message along any of its channels at time slot t, the corresponding information parcels are immediately added to its upload buffer. Each parcel added to the upload buffer of p_i has an intended recipient p_j. At the end of timeslot t, S parcels (or x parcels if the buffer presently only holds $x < S$ parcels) are removed from the upload buffer in a FIFO manner, and then appear on the download buffers of their intended recipients at $t+\Delta$. At the beginning of a time slot, (up to) S parcels are removed from the download buffer in a FIFO manner, and those parcels are then 'received' by the processor, together with information specifying the sender of each parcel.

Modeling connections to clients. In a state machine replication protocol, the n processors receive transactions from external clients. Thus, the processors utilize their bandwidth to also receive transactions from these clients. To model this cleanly, we suppose that every logical processor consists of two physical processors, a *client* processor and a *consensus* processor, which are connected to each other using an infinite bandwidth and zero delay channel. Communication channels between logical processors are between their respective consensus processors, and the upload and download buffers of those processors impact the rate at which information can be sent along these channels between consensus processors. The clients send data to client processors and we will suppose that the client processors in the system receive transactions at a total (combined) rate of D parcels per time slot.[8] The transactions received through the download buffer of the client processor are sent to the consensus processor through the infinite bandwidth connection, and thus the consensus processors receive this information at the combined rate of D parcels per time slot.

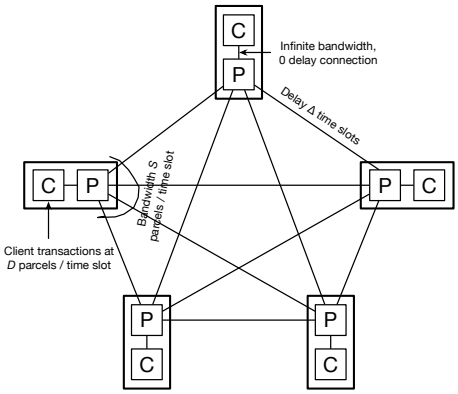

Figure 3: Depicting a 5-processor system. Each processor involves a client processor (denoted C) and consensus processor (denoted P).

The single-sender and multi-sender settings. The idea behind DAG-based protocols is that each processor should propose their own blocks of transactions. Typically, this can result in some complications. For instance, different processors may end up transmitting the same data, thus resulting in repetitive work, or incompatible transactions may be transmitted by different processors. In this work, we ignore such concerns and assume that all of the transmitted data is useful. In the multi-sender setting, for example, we suppose that the D transaction parcels arriving at client processors per time slot are (somehow) evenly divided between the client processors, so that each receives D/n transaction parcels per time slot:

> In the *single-sender* setting, a designated leader's client processor receives D transaction parcels at each timeslot (other processors do not receive transaction parcels at their client processors). In the *multi-sender* setting, each processor receives D/n transaction parcels at its client processor at each timeslot. We suppose that all transactions are unique.

Latency. The notion of 'delivery' varies by protocol (e.g., receiving a full block in Best-effort Broadcast, or finalizing in SMR) and will be made precise in each context. A protocol has latency r if in every execution, any data parcel received by a correct processor at time t is delivered by all correct processors by $t + r$. We analyze worst-case latency in synchronous periods, when processors behave correctly.

6. For simplicity, each parcel can be viewed as a single bit, though allowing variable-sized parcels can sometimes be useful.

7. In practice, delays between processors in a geo-distributed setting are non-uniform, e.g., two processors on the U.S. east coast experience lower latency than a processor in the U.S. communicating with one in Asia. Nevertheless, we assume uniform delays for simplicity.

8. Whether or not the download buffer of *client* processors is rate limited will not meaningfully impact our analysis.

3. Best-Effort Broadcast

Our ultimate aim is to use our model to compare latency for state-of-the-art SMR protocols. As a stepping stone, we first consider latency for a number of primitives such as BeB, CB, and RB. In each case, we consider the 'multi-shot' version (in which one must reach a sequence of consensus decisions, rather than a single decision), since SMR protocols that use these primitives do so in the multi-shot setting.

In this section, we consider BeB protocols. Although these protocols are simple, we spend significant time analyzing them, since doing so serves as a good introduction to the methods required to analyze more complicated protocols.

3.1. Best-effort Broadcast with a single sender

There is a designated processor p_ℓ. The following instructions are for p_i.
at time slot 0 **do**
  $\mathsf{block\text{-}txns} = \{\}$ ▷ $\{\}$ denotes the empty sequence
  $\mathsf{sndbuf} = \{\}$ ▷ sndbuf is a FIFO queue distinct from the upload buffer
  $\mathsf{recv\text{-}block} = \{\}$
at each time slot t **do**
  if $p_i = p_\ell$ **then**
    p_i collects *txns*, which is all data parcels received by the client processor but not yet added to sndbuf
    $\mathsf{block\text{-}txns} = \mathsf{block\text{-}txns} \cup \mathit{txns}$
    $\mathsf{sndbuf} = \mathsf{sndbuf}||\mathit{txns}$
    Dequeue the first S/n parcels in sndbuf and add each to the upload buffer n times, once for each recipient ▷ Send some parcels
    if $|\mathsf{block\text{-}txns}| \geq \frac{\alpha}{1-\alpha} M$ **then** ▷ M denotes the total size of metadata ▷ α is as specified in Section 3.1
      Set $\mathrm{md}^{\mathrm{de}}$ = *dependent* metadata for the current block
      Set $\mathrm{md}^{\mathrm{in}}$ = *independent* metadata for the next block ▷ dependent and independent metadata as specified in Section 3.1
      $\mathsf{sndbuf} = \mathsf{sndbuf}||\mathrm{md}^{\mathrm{de}}||\mathrm{md}^{\mathrm{in}}$
      $\mathsf{block\text{-}txns} = \{\}$
  upon receiving parcels m from p_ℓ **do**
    $\mathsf{recv\text{-}block} = \mathsf{recv\text{-}block}||m$
    if $|\mathsf{recv\text{-}block}| \geq \frac{M}{1-\alpha}$ **then** ▷ recv-block includes metadata
      Deliver recv-block as $(\mathrm{md}^{\mathrm{in}}, \mathit{txns}, \mathrm{md}^{\mathrm{de}})$ received from p_ℓ
      $\mathsf{recv\text{-}block} = \{\}$

Figure 4: Best-effort Broadcast by a single designated processor

The first primitive we consider is a form of BeB in the single-sender setting. A designated 'leader' p_ℓ receives D data parcels per time slot at its client processor and sends parcels to all processors. Our goal is to consider the rate at which p_ℓ can push data to all others, in a context where data should be included in 'blocks' that must also contain metadata, and the bound this puts on throughput and latency: to specify latency, processors 'deliver' a transaction when they receive a full block containing that transaction. We initially suppose that p_ℓ sends data to others in a straightforward fashion, without using techniques such as erasure coding.

Data parcel terminology. We use the following terminology. Data parcels arriving at the client processor are called *transactions*. A block is made up of *block data parcels*: these are either transactions or *metadata parcels*. *Addressed parcels* are those in the upload/download buffers of consensus processors. These are either *addressed transactions* or *addressed metadata*.

Protocol overview. Figure 4 describes the protocol.[9] The goal is for the leader to produce a block of transactions and send it to all processors along with some metadata, while minimizing latency. This task is complicated by the fact that we must consider two types of metadata:

- *Dependent metadata*, such as p_ℓ's signature on the block, which cannot be determined prior to deciding the transactions included in the block.
- *Independent metadata* that can be determined before deciding which transactions are included in the block. For example, p_ℓ may be able to determine an appropriate hash pointer to a previous block before the transactions are decided.

We let M denote the total size of the metadata for each block, while the size of dependent and independent metadata are denoted M^{de} and M^{in} respectively (we assume these sizes are fixed and do not vary between blocks). As we will see, it is important to distinguish these different forms of metadata because they have different impacts on latency.

To specify the block, the leader maintains a local variable called sndbuf (not to be confused with the upload buffer), and adds transactions to sndbuf.[10] A crucial distinction between sndbuf and the upload buffer is that the latter needs to contain a distinct copy of each block data parcel for each recipient. At each timeslot, p_ℓ therefore makes n copies[11] of some of the parcels in sndbuf and adds these addressed parcels to its upload buffer (adding parcels to the upload buffer constitutes the decision to 'send' those parcels). Once the number of transactions added to sndbuf reaches a predetermined size, it computes the dependent metadata for the *current* block and the independent metadata for the *next* block and appends that data to sndbuf. Note that independent metadata is sent before all other block data, while dependent metadata is placed at the end of the block. On the receiver's end, every processor receives data at each timeslot until it has received a complete block along with the associated metadata.

Understanding the protocol parameterization. Recall that D transactions arrive at the leader's client processor per time slot. As a first observation, we note a simple upper

9. For simplicity of presentation, we ignore integer rounding issues (e.g., when S/n is not an integer) in our protocol description and analysis.

10. Throughout the paper, we use $x||y$ to denote x concatenated with y. For example, "$\mathsf{sndbuf} = \mathsf{sndbuf}||y$" means appending y to sndbuf.

11. For simplicity, we assume that p_ℓ sends data to all processors, including itself.

bound on values of D for which bounded latency is possible: since the leader has a bandwidth of S and since each transaction arriving at the client processor needs to be sent to n processors, if $Dn > S$ then the required data cannot be taken from the upload buffer of the leader as fast it is arriving at its client processor. This means that the latency for subsequent transactions will grow in an unbounded fashion. For the remainder of Section 3.1, we therefore assume that $D \leq S/n$. In particular, suppose $D = \alpha S/n$ for $\alpha \in (0, 1]$. Observe that if $\alpha = 1$, the leader can indeed disseminate the arriving transactions but does not have additional bandwidth to send any metadata. We will see that, for any $\alpha < 1$, setting the number of transactions included in each block to be $\frac{\alpha}{1-\alpha}M$ allows the leader to send metadata along with the transactions, while also minimizing latency.

We make the following claim:

Claim 1. *Suppose all processors are correct. Let M be the total size of the block metadata, while M^{de} is the size of the dependent metadata. If $D = \alpha S/n$ for $\alpha \in (0, 1)$, then latency for the protocol described in Figure 4 is:*

$$\frac{(M+(1-\alpha)M^{de})n}{(1-\alpha)S} + \Delta \quad = \frac{(M+(1-(nD/S))M^{de})n}{S-nD} + \Delta.$$

Proof. Consider first what happens between timeslot 0 and the first timeslot, t_1 say, at which p_ℓ adds metadata to sndbuf. At each timeslot in this interval, less than S/n transactions arrive at the client processor of p_ℓ (since $\alpha < 1$), and are immediately added to sndbuf. Each is then added to p_ℓ's upload buffer n times (once for each recipient), with the upload buffer emptied of addressed transactions by the end of the timeslot. It follows that, by the end of time slot t_1, sndbuf does not contain any transactions and that the leader's upload buffer does not contain any addressed transactions.

Now suppose that, at some timeslot t, the leader completes a block, i.e., it adds the dependent metadata for some block as well as the independent metadata for the next block to sndbuf. Suppose (inductively) that, by the end of timeslot t, sndbuf does not contain any transactions and that the leader's upload buffer does not contain any addressed transactions (i.e. all such parcels added to these buffers have been removed). Recall that the leader receives $D = \alpha S/n$ transactions at its client processor every time slot. This means a set of $B := \frac{\alpha}{1-\alpha}M$ transactions is received in time:

$$\frac{B}{D} = \frac{Bn}{\alpha S} = \frac{Mn}{(1-\alpha)S}.$$

Note that the metadata added to sndbuf at t is of size M and that $B + M = \frac{M}{1-\alpha}$. From each of these block data parcels added to sndbuf, the leader needs to form n addressed data parcels to add to the upload buffer. S parcels are removed from the upload buffer of the leader at each timeslot, corresponding to S/n parcels from sndbuf. To remove the addressed versions of the $B + M = \frac{M}{1-\alpha}$ block parcels from the upload buffer therefore takes $\frac{Mn}{(1-\alpha)S}$ time, which is precisely the amount of time it takes for the B transactions to arrive at the client processor of the leader. This means that, when the leader completes the next block, b say, it will once again hold that sndbuf does not contain any transactions and that the upload buffer does not contain any addressed transactions. Let t' be the time at which the leader completes b, i.e., adds the dependent metadata for b (together with the independent metadata for the next block) to sendbuffer.

Recall that the latency is the (longest possible) time gap between a transaction arriving at the client processor of the leader and all correct processors having received a full block containing the transaction. To compute the latency, we have to determine the time between t, when transactions placed in b first start arriving at the client processor of the leader, and the time t'' when all correct processors have received b. Note that, although the leader placed dependent metadata in sendbuffer at t, this was the dependent metadata for the *previous* block. Once the leader completes b at t', to determine t'' we still have to consider the extra time taken to remove the dependent metadata for b from the upload buffer, together with the message delay Δ (noting that the download buffers of other processors are never a bottleneck in this analysis). Removing the addressed metadata md^{de} for b from the upload buffer requires $\frac{M^{\text{de}}n}{S}$ time slots. As claimed, this gives a latency of:

$$\frac{Mn}{(1-\alpha)S} + \frac{M^{\text{de}}n}{S} + \Delta = \frac{(M+(1-\alpha)M^{\text{de}})n}{(1-\alpha)S} + \Delta.$$

□

Analysis. If $B < \frac{\alpha}{1-\alpha}M$, then latency will be unbounded. On the other hand, any $B \geq \frac{\alpha}{1-\alpha}M$ suffices for bounded latency, while $B = \frac{\alpha}{1-\alpha}M$ minimizes the latency. As α tends to 1, block size and latency tend to infinity.

When α is fairly small, latency is roughly $(M + M^{\text{de}})n/S + \Delta$. If n is small enough or S is large enough that $(M + M^{\text{de}})n/S$ is small compared to Δ, then latency is dominated by Δ. As n becomes large, the former term dominates and latency becomes roughly linear in n.

3.2. Best-effort Broadcast by all processors

We next consider a related primitive where every processor sends parcels to all processors in the multi-sender setting. The dissemination process is thereby parallelized, potentially leading to a more even use of the available bandwidth. Each processor needs to transmit D/n parcels per timeslot. Our goal is to compute the latency for pushing these parcels to all processors. Similar to the case with a single sender, there is an upper bound on the values of D for which bounded latency is possible. In particular, there is now a latency bottleneck at $D = S$ (as opposed to S/n). If $D > S$, each processor cannot download/upload data from/to other processors at the rate at which it is arriving.

The precise description of the protocol appears in Figure 10, and the proof of the following claim is deferred to Appendix A.

Claim 2. *Suppose $D = \alpha S$ for $\alpha \in (0,1)$ and that all processors are correct. Then latency for the protocol in Figure 10 is:*

$$\frac{(M + (1-\alpha)M^{de})n}{(1-\alpha)S} + \Delta \quad = \frac{(M + (1-\frac{D}{S})M^{de})n}{S-D} + \Delta.$$

Analysis. The principal difference between the results for the single-sender setting and the multi-sender setting is that the latency bottleneck is increased by a factor n for the latter setting. In Appendix D, we also establish a sense in which the protocols of Figures 4 and 10 are optimal: there is a precise sense in which each has the lowest possible latency amongst BeB protocols (for their respective setting) that 'do not use techniques like erasure coding'.

4. RB, CB, and RB with erasure coding

The requirement for CB is that no two correct processors should deliver different values (blocks from the sender in our case), and that correct processors should deliver the sender's block if the latter is correct. RB has the extra requirement that, if any correct processor delivers a block from the sender, then all correct processors must do so. Typically, CB is executed by sending the block to all processors and then delivering the block upon receiving a quorum of votes that ensure both uniqueness and *data availability*, i.e., some correct processor has the block.

In this section, we consider latency for standard forms of RB, CB, and RB with erasure coding. To conserve space, we explain RB with a single sender, and only state other results here. Appendices E–K describe all protocols and formally establish all remaining results that are later used in Section 5, when we consider SMR protocols. When analyzing multi-sender protocols, we suppose processors are synchronized, i.e., begin the protocol execution at the same time as each other.

Reliable Broadast. In the single-sender setting, a designated leader receives D transaction parcels per time slot and sends blocks to all others using a version of Bracha's broadcast [21], assuming $f < n/3$ faulty processes. The protocol is shown in Figure 16.

In our analysis, we make the following simplification: if a block b contains B transaction parcels and metadata of size M, then we also suppose that the message (ECHO, d, b) is of size $B + M$. We suppose that *votes*, i.e. messages of the form $(\text{VOTE}, d, H(b))$, are of a fixed size λ.

Claim 3. *Suppose all processors are correct. If $S/Dn > 2$ and the block size B is set to minimize latency, then the latency of the protocol of Figure 16 is:*

$$\left(\frac{(4M + 2\lambda)n}{S} + 6\Delta\right)\left(1 + \frac{3/2}{(S/Dn) - 2}\right).$$

Proof. Suppose B transactions are included in each block and that, at some time slot t, the leader delivers a block of depth $d-1$ and begins sending transactions for the next block b of depth d. All processors will then deliver b by time $t' := t + 2(B+M)n/S + n\lambda/S + 3\Delta$ at the earliest. Since $D(2(B+M)n/S + n\lambda/S + 3\Delta)$ transactions will arrive at the client processor of the leader between t and t', if latency is to be bounded we require:

$$B \geq \frac{D(2(B+M)n + n\lambda + 3\Delta S)}{S}.$$

From this it follows that: $B \geq \frac{D(2nM + n\lambda + 3\Delta S)}{S - 2nD}$. Set:

$$B := \frac{D(2nM + n\lambda + 3\Delta S)}{S - 2nD}. \tag{1}$$

Consider next what happens between timeslot 0 and the first timeslot, t_1 say, at which p_ℓ adds metadata to sndbuf. At each timeslot in this interval, less than $S/2n$ transactions arrive at the client processor of p_ℓ (since $S/Dn > 2$), and are immediately added to sndbuf. Each is then added to p_ℓ's upload buffer n times (once for each recipient), with the upload buffer emptied of addressed transactions by the end of the timeslot. It follows that, by the end of time slot t_1, sndbuf does not contain any transactions and that the leader's upload buffer does not contain any addressed transactions.

Now suppose that, at some timeslot t, the leader completes a block, i.e., while its local value send = true, it adds the metadata for some block to its sndbuf. Suppose (inductively) that, by the end of timeslot t, its sndbuf does not contain any transactions and that its upload buffer does not contain any addressed transactions (i.e., all such parcels added to these buffers have been removed). All correct processors will then deliver the next block at time:

$$\begin{aligned} t' &:= t + \frac{Mn}{S} + \frac{B+M}{S} + \frac{\lambda}{S} + 3\Delta \\ &= t + B(S-2nD)/(DS) + Bn/S. \end{aligned}$$

To see that the second equality above holds, feed the expression for B in (1) into the expression on the right. In the interval $(t, t']$, the leader p_ℓ will therefore receive $B(S-2nD)/S + BnD/S$ many transaction parcels, which will be immediately added to sndbuf at t'. To complete the block, the leader requires a further

$$B - B(S - 2nD - nD)/S = nBD/S$$

transaction parcels, which will arrive in time nB/S, which is exactly the time that p_ℓ requires to clear sndbuf of all transactions. It therefore holds that when the leader completes the next block at $t'' := t' + nB/S$, its sndbuf does not contain any transactions and that its upload buffer does not contain any addressed transactions.

To calculate the latency, we have to consider the time from when transactions start building up at t until the next block (which the leader starts sending at t') is delivered by all correct processors. This is:

$$\frac{(3B + 4M + 2\lambda)n}{S} + 6\Delta.$$

Substituting in the value for B in (1) gives the latency as claimed. Note also that increasing B increases latency

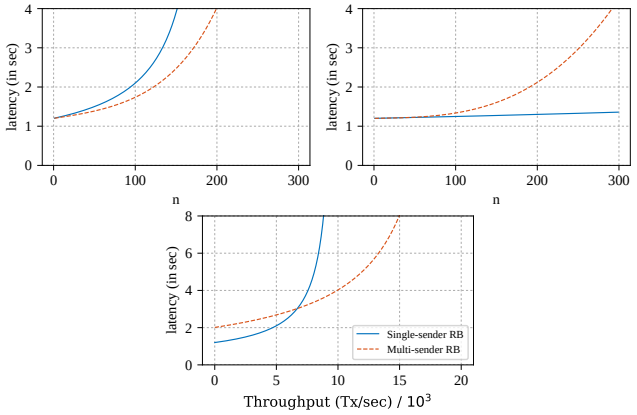

Figure 5: Latency for RB: parameters are explained in Section 4.

according to the calculations above, so that B as specified in (1) minimizes latency. □

As formally established in Appendix H, after lower order terms are removed, the corresponding latency in the multi-sender setting if $Dn < S$ is:

$$\left(\frac{2(M+\lambda)n^2}{S} + 6\Delta\right)\left(1 + \frac{1}{(S/Dn) - 1}\right). \quad (2)$$

So, for RB, the factor n difference in maximum throughput disappears: the corresponding bottlenecks are $2Dn = S$ and $Dn = S$, and so differ only by a factor of 2. This is caused by the need for processors to 'echo' the blocks of all others in the multi-sender setting. The factor of 2 arises because, in the single-sender case, there are *two* rounds which are equally expensive in terms of latency: first the leader sends their block to all, and then all others echo it to all. In the multi-sender case, the single round which dominates in terms of latency cost is that in which each processor has to echo-to-all all blocks produced by other processors.[12]

To illustrate these results, we consider parameter values which are the same as in Section 1, except that we set $S = 10$ Gbps. The first graph in Figure 5 supposes the incoming transaction rate D is 10000 transactions per second and shows the resulting latency (in seconds) as a function of n. The second graph (on the right) supposes the incoming transaction rate D is 1000 transactions per second and shows the resulting latency as a function of n. The lower graph fixes $n = 200$ and shows the resulting latency as a function of the number of incoming transactions per second divided by 1000: one can see that the latency bottleneck in the single-sender case appears at 10000 transactions per second.

Consistent Broadcast. We consider a standard form of CB: see Appendices E and F for further details. In the single-sender setting, if votes are of size λ and block metadata is of size M, and if $Dn < S$, latency is:

$$\left(\frac{2(M+\lambda)n}{S} + 4\Delta\right)\left(1 + \frac{1}{2((S/Dn) - 1)}\right).$$

For CB in the multi-sender setting, if $D < S$ the corresponding latency is:

$$\left(\frac{2Mn + 2\lambda n^2}{S} + 4\Delta\right)\left(1 + \frac{1}{2((S/D) - 1)}\right). \quad (3)$$

The expressions above are formally established in Appendices E and F. Once again, the main difference between these two results is that the latter has a latency bottleneck (maximum throughput) that is higher by a factor of n. However, one should not think that this factor n difference applies to all primitives. Indeed, we have seen that this is not so for RB.

RB with erasure coding. Appendices I–K include a number of results for RB with erasure coding. Here, we just highlight a result that establishes latency for a form of RB with erasure coding and pipelining in the single-sender setting, which is the same as that used by (some versions of) DispersedSimplex [20], and which gives the following latency if $3D < S$:

$$\left(\frac{6M+2n\lambda(\log(n)+4)}{S}\right) \cdot \left(1 + \frac{1}{(S/3D)-1}\right) + 2n\lambda/S + 3\Delta. \quad (4)$$

5. Latency for SMR protocols

The difficulty of making apples-to-apples comparisons. For a number of reasons, making a direct comparison between protocols for the single-sender and multi-sender settings is complicated. To start with, the two settings make inherently different assumptions. In some contexts, one may consider the assumption that transactions are divided evenly between processors without repetition in the multi-sender setting as generous: it is not immediately clear how such a division is to be achieved in a decentralized fashion, and whether doing so will induce substantial extra latency [22]. In other contexts, such a partition of transactions (or an approximation to it) is likely to arise naturally, while relaying all transactions to a leader may induce extra latency [23].

Different protocols also give different guarantees in terms of data availability. In comparing latency for versions of Sailfish, the version using CB is seen to have lower latency than others, but CB gives weaker guarantees than RB in terms of ensuring that correct processors receive the necessary blocks. These weaker guarantees may be abused by Byzantine actors. Further complications arise from the fact that, in the single-sender setting, the use of erasure coding is the only method we are aware of which allows for a latency bottleneck at $D = \Theta(S)$ (rather than $Dn = \Theta(S)$). In reality, the use of erasure coding may be computationally expensive. Gossip networks are an alternative approach, but we leave an analysis of this method to future work.

Since it is difficult to make an entirely apples-to-apples comparison, we focus on comparing the best (i.e., lowest

12. The factor of 2 can also be eliminated by using *pipelining* techniques in the single sender case: dropping the requirement for the leader to echo their own blocks, they may immediately progress to broadcasting their next block while others echo the current one.

latency) protocols for each of the single-sender and multi-sender settings, while bearing in mind that there remain hidden trade-offs if one just considers the corresponding latency figures.

5.1. Latency for DAG-based SMR protocols

In this section, we consider latency for standard 'certified' DAG-based SMR protocols in the multi-sender setting and assume the reader is familiar with such protocols. Each processor now *delivers* a transaction upon adding a block containing that transaction to its finalized ledger.

In fact, analyzing latency for such protocols is generally rather straightforward, given the analysis we have already carried out in previous sections: calculating latency now amounts to counting the number of rounds of CB/RB (or possibly RB with erasure coding, whichever is employed by the protocol in question for the task of block propagation) that are required to finalize each block.

The example of Sailfish. To make things concrete, we focus on the example of Sailfish [12] with CB as the method of block propagation, and assume familiarity with that protocol in what follows. However, the analysis we carry out here is easily adapted to deal with other DAG-based protocols.

As with many other DAG-based protocols, one complexity is that the latency incurred by a given transaction may depend on the position of the corresponding block in the DAG. Adopting a standard technique, we suppose that, as well as pointing to $n - f$ blocks from the previous layer, each block for layer d can include pointers to f blocks from layers $< d - 1$. During synchrony, and if processors are correct, then each block for layer d will be in the 'history' of (i.e., will be transitively pointed to by) each block for layer $d + 2$.

Sailfish with CB as the underlying primitive. We simplify the analysis by assuming that all processors are correct and are 'in-sync', i.e., that all processors begin construction of each layer of the DAG simultaneously. To calculate the latency, we consider transactions included in a block b produced by p_i in layer d of the DAG. As already specified by Equation (3), the maximum delay between receipt of a transaction in b and the point at which all processors CB-deliver b is:

$$\left(\frac{2Mn + 2\lambda n^2}{S} + 4\Delta\right)\left(1 + \frac{1}{2((S/D) - 1)}\right)$$

We then have to consider the latency induced by the time to CB-deliver blocks in layers $d + 1$ and $d + 2$. However, for each of these layers we *do not* have to consider any equivalent of the initial delay between receipt of a transaction in b and the time at which p_i starts sending b. For each of these layers the contribution to latency is:

$$\left(\frac{Mn + \lambda n^2}{S} + 2\Delta\right)\left(1 + \frac{1}{(S/D) - 1}\right)$$

.

Since all transactions in b will be finalized by all correct processors when they CB-deliver a leader block for layer $d + 2$ (or, perhaps, $d + 1$) and then receive a first message from the leader of the next layer, this gives total latency:

$$\left(\tfrac{4(Mn+\lambda n^2)}{S} + 8\Delta\right)\left(1 + \tfrac{3}{4((S/D)-1)}\right) + C(B + M) + \Delta.$$

Substituting in the optimal value for B calculated in Appendix F, this is:

$$\left(\frac{4(Mn + \lambda n^2)}{S} + 8\Delta\right)\left(1 + \frac{1}{(S/D) - 1}\right) + \frac{Mn}{S} + \Delta.$$

If $M = n\lambda$, this gives the expression given in Section 1:

$$\left(\frac{9\lambda n^2}{S} + 9\Delta\right)\left(1 + \frac{8}{9((S/D) - 1)}\right). \tag{5}$$

Sailfish with RB as the underlying primitive. The calculations here are entirely analogous to those for CB, but use instead the analysis of Section H. A similar analysis now shows the latency to be:

$$\left(\tfrac{4(M+\lambda)n^2}{S} + 12\Delta\right)\left(1 + \tfrac{5}{4((S/Dn)-1)}\right) + Mn/S + \Delta.$$

The significant observation here is that the use of RB (rather than CB) reduces the maximum throughput from $D = S$ to $D = S/n$.

5.2. Latency for SMR protocols using erasure coding and pipelining in the single-sender setting

To make things concrete, we consider DispersedSimplex. We assume familiarity with that protocol in what follows and consider latency in the context of a stable leader, assuming all processors are correct.

In fact, analyzing DispersedSimplex in this setting is very simple given our analysis for RB with erasure coding and pipelining in the single-sender setting: when processors behave correctly, and during synchrony, latency for DispersedSimplex is exactly as given by Equation 4:

$$\left(\tfrac{6M+2n\lambda(\log(n)+4)}{S}\right)\left(1 + \tfrac{1}{(S/3D)-1}\right) + 2n\lambda/S + 3\Delta. \tag{6}$$

Of course, the instructions for DispersedSimplex must also allow for the sending of messages, such as *complaint certificates*, in the case that the leader is faulty or message delivery is not reliable. However, these instructions do not impact latency in the good case.

5.3. Latency for Tendermint and HotStuff

The same methods used to calculate latency for CB and RB in Appendices F and H can also be used to calculate latency for Tendermint and HotStuff. These calculations are demonstrated in Appendices L and M. For Tendermint, latency is:

$$\tfrac{(6\lambda n+2Mn)/S+10\Delta}{(S/Dn)-1} + \tfrac{(5\lambda+2M)n}{S} + 8\Delta. \tag{7}$$

The corresponding expression for HotStuff is:

$$\frac{(14\lambda n+2Mn)/S+16\Delta}{(S/Dn)-1} + \frac{(13\lambda+2M)n}{S} + 15\Delta. \quad (8)$$

Comparing with Equation (23), we see that latency for HotStuff is strictly greater than latency for Tendermint, for all parameter values.

5.4. Comparing latency for SMR protocols

Section 5.3 demonstrates something which has been intuitively understood by many practitioners: the reduced communication complexity of HotStuff compared to Tendermint does not reduce latency, because the leader acts as a bottleneck. The advantage of DAG-based protocols over standard single-sender protocols that do not use erasure coding is that the former have a maximum throughput that is greater by roughly a factor of n.

However, comparisons between DAG-based protocols and single-sender protocols like DispersedSimplex that use erasure coding are more subtle. While the latter protocols may have lower latency over many regimes, the advantage of the former protocols is that they have a maximum throughput which is greater by a constant factor. With standard erasure coding techniques, this factor is 3, since a leader propagating a block of size $B+M$ bits must send a total of roughly $3(B+M)$ bits of data to all processors combined. We note that the most recent version of DispersedSimplex also includes an optimization that can reduce this factor 3 to a factor 1.5 in the optimistic case and with a stable leader.

Figures 1 and 2 demonstrate some of the trade-offs between these various protocols.

6. Evaluation

We implement and evaluate both single-sender and multi-sender BeB primitives, as well as variants of the Sailfish and DispersedSimplex SMR protocols, and compare their empirical performance against our theoretical model. All implementations are written in Rust. The source code for single-sender BeB is available at [24] and multi-sender BeB at [25]. Similarly, the source code for Sailfish is available at [26] and DispersedSimplex at [27].

Experimental setup. We conduct our experiments on the Google Cloud Platform (GCP), deploying all nodes within a single region to ensure consistent baseline network conditions. Each node runs on an e2-standard-4 instance [28], equipped with 4 vCPUs, 16 GB of memory, and up to 8 Gbps network bandwidth, running Ubuntu 22.04. The native ping latency between nodes is below 1 ms. To emulate realistic network conditions in a controlled manner, we use the Linux traffic control (tc) utility [29], which supports fine-grained configuration of delay, bandwidth, and packet loss. We configure tc to enforce a uniform 100 ms one-way delay between all nodes and cap each node's bandwidth at 100 Mbps. This setup provides reproducible experiments under realistic network constraints.

We use TCP connections for data transfer between nodes. TCP employs congestion control, which initially limits the number of in-flight packets (typically to around 10) before additional packets can be sent. This results in higher latency at the start of a connection. As more data is transmitted, the congestion window automatically expands, enabling lower latency and higher throughput. Similarly, TCP send and receive buffers start with small default values to reduce memory usage and are gradually increased as the connection progresses, which can also introduce initial latency. To avoid these startup effects, we ensure that all TCP connections are properly warmed up (i.e., the congestion window and buffer sizes have scaled to stable values) before running our benchmarks.

6.1. BeB primitives.

Implementation details. Our implementations of the single-sender and multi-sender BeB protocols differ from the specifications in Figures 4 and 10. In those models, S parcels are assumed to be sent to the network in every time slot. However, in practice, the network may not be able to transmit S parcels within a single time slot, and injecting additional parcels before previous ones have been fully transmitted can introduce additional latency. Moreover, to the best of our knowledge, the operating system does not expose information about when data has actually been uploaded to the network. Time-based event triggers provided by the OS are also imprecise, further complicating implementation.

To address these challenges and simplify implementation, we upload the entire block of size $\frac{\alpha M}{1-\alpha}$ at once and then wait for an estimated clearing time (provided as an input parameter) before transmitting metadata: of course, the theoretical analysis is adjusted accordingly. If this estimate is too small, metadata may be sent before previous transmissions complete, increasing latency. A separate estimate is also needed for metadata upload time. Consequently, we experiment with multiple values for these estimates to identify those that yield latency close to the theoretical predictions for the modified protocol.

Performance discussion. We evaluate the performance of the BeB primitives across system sizes of 10, 20, 30, and 40 nodes. In all experiments, we set $D = 250{,}000$ Bps, $\mathrm{md}^{\mathrm{de}} = 12{,}500$ bytes, and $\mathrm{md}^{\mathrm{in}} = 25{,}000$ bytes, and run each experiment for 120 seconds. In these experiments, we rely on the time taken by the last party to deliver the block and its metadata to compute the latency and we report the median latency during the run. The results for single-sender BeB are shown in the left graph of Figure 6, with the results for multi-sender BeB on the right.

As shown in Figure 6, the performance of the single-sender BeB closely matches our theoretical model. For instance, at $n = 40$, the observed latency is only about 5% higher than predicted. These minor deviations are primarily due to TCP communication overhead and slight discrepancies between the actual network bandwidth and the limits

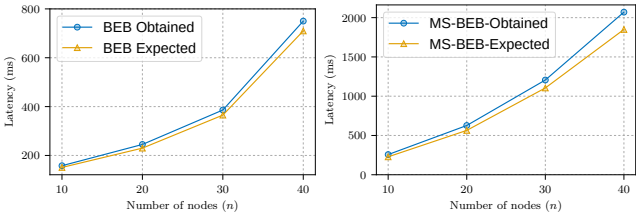

Figure 6: Obtained vs. expected latency for single and multi-sender BeB

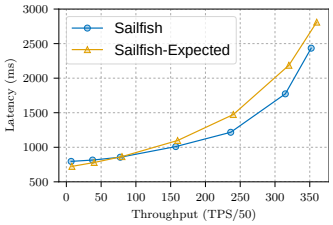

Figure 7: Throughput vs. latency of Sailfish at $n = 40$.

configured using the tc utility. Overall, these results validate the accuracy of our theoretical analysis. Importantly, the performance patterns are the same.

The performance of the multi-sender BeB deviates slightly more from the theoretical prediction than the single-sender case. For example, at $n = 40$, the observed latency is about 12% higher than predicted. This deviation primarily arises from multiple concurrent senders transmitting data to the same recipient simultaneously, coupled with the inherent unpredictability in the scheduling behavior of the underlying network layer. Nevertheless, the overall pattern of latency growth closely follows our theoretical prediction, thereby validating the accuracy of our analytical model.

6.2. SMR Protocols

Implementation details. We simplify the implementations of both Sailfish and DispersedSimplex. In our Sailfish implementation, we employ the CB primitive to disseminate proposed blocks and wait for all n vertices in each layer, rather than $n - f$ as specified in the original protocol. Furthermore, incoming transactions are first accumulated in a buffer and proposed collectively in the next block proposal. This differs from the design in Figure 15, where incoming transactions can be included in the block currently being disseminated: the theoretical analysis of this modified implementation is adapted accordingly.

For DispersedSimplex, we implement a non-pipelined variant in which the leader waits to obtain a quorum certificate for the previous block before issuing a new proposal. Similar to Sailfish, the leader accumulates incoming transactions in a buffer and proposes them collectively in the next block proposal. Furthermore, we omit actual erasure coding and instead transmit data of equivalent size to an erasure-coded chunk to avoid the computational overheads of encoding, decoding, and chunk verification. Additionally, both SMR implementations exclude digital signatures to eliminate cryptographic computation overhead, as this cost is not accounted for in our theoretical model.

Performance discussion. We vary D to obtain varying throughput and latency for these protocols. The performance comparison for Sailfish is shown in Figure 7. Interestingly, the observed latency in our implementation is generally lower than the theoretical prediction. This discrepancy likely arises because blocks are proposed in bursts, followed by idle periods during which only minimal traffic (e.g., vote messages) is exchanged. (This behavior differs from the BeB primitives, where data is continuously streamed, keeping both the upload and download channels consistently active.) The Linux tc subsystem regulates outgoing bandwidth using token-bucket–based schedulers, which permit short transmission bursts above the configured steady-state rate when sufficient tokens accumulate during idle periods. The scheduler maintains a token bucket that refills at a fixed rate, representing available transmission capacity. When the network is idle, excess tokens accumulate, enabling brief bursts that temporarily exceed the nominal bandwidth until the bucket is depleted [30].

Despite these effects, the overall trend (latency increasing with throughput) closely aligns with our theoretical analysis, reinforcing the validity of our model.

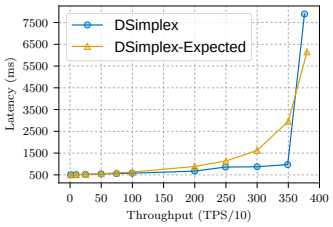

Figure 8: Throughput vs. latency of Dispersed Simplex at $n = 40$.

The performance comparison for DispersedSimplex is shown in Figure 8. The observed latency in our implementation is generally lower than the theoretical prediction. This occurs because blocks are transmitted in bursts, leaving the network idle on the receiver's end before new proposals arrive. As discussed earlier, this allows temporary bandwidth surges during bursts, resulting in lower latency. However, when D approaches S, the leader's bandwidth becomes fully utilized, eliminating burst transmissions. In this regime, the observed latency exceeds the theoretical prediction due to additional overheads such as storage and system-level delays which are not captured by our model.

7. Related Work

Latency of consensus protocols. In the existing literature, latency is often measured in terms of the number of 'rounds' of communication required for asynchronous and partially synchronous protocols (and sometimes synchronous protocols too), and in terms of the pessimistic network delay Δ for synchronous protocols [31]. This is because, under asynchrony, the network delay can be arbitrary, and so one cannot provide a bound on latency in terms of Δ.

Many theoretical studies have focused on the worst-case latency (given arbitrary behaviour by a bounded adversary). For Byzantine broadcast, the worst-case number of rounds required is $f + 1$ to tolerate up to f Byzantine faults [32]. Intuitively, in many protocols, this is because "leaders" may behave maliciously, e.g., not send any messages, and so need to be changed until one has an honest leader. An alternative latency measure is the 'expected (round) latency'. At a high level, protocols may select leaders uniformly at random, and so, if the correct parties are at least a constant fraction of the total number of parties, the expected latency is $O(1)$ rounds [33].

In addition, many prior works do not take into consideration the amount of data sent when describing the latency of the protocol. For instance, we say Byzantine reliable broadcast and partial synchrony broadcast have a latency of 2 rounds and 3 rounds respectively [31]; for larger data sizes, the rounds may be longer in duration but the metric does not account for it. Even when expressed in Δ, they assume that the same Δ parameter is sufficient to send any amount of data. This is reasonable in scenarios where small fixed size inputs are agreed upon. An exception to this is [16] in which Bagaria et al. describe a model (further developed in [17], [34], [35]) that is tailored to the analysis of longest chain protocols that transmit all messages using a gossip network. A basic similarity with our approach is that they consider a notion of 'network capacity' C and suppose that sending a block containing B transactions should take time $\Delta := B/C + D$ for some fixed delay D. However, their model then divides the execution into discrete rounds of length Δ (somewhat akin to the analysis in [36]). Any block sent during round r arrives at the beginning of round $r + 1$. To limit the number of blocks that can be sent in each round, they consider an 'environment'. All messages are sent to and delivered by the environment, and the environment is allowed to process at most $C\Delta$ transactions in each round. The 'capacity' therefore acts as a network-wide limit on the ability to pass messages/transactions from each round to the next and the model does not allow for an analysis of the extent to which specific communication channels between pairs of processors may act as a bottleneck. Another crucial distinction is that, to simplify calculations, they carry out a sort of 'mean-field' approximation in the limit of large n (and assume that the network is somehow able to deal with delivering transactions to infinitely many processors in finite time). By contrast, the impact of n on latency is a focus of our analysis. The approach described by our model is therefore much more granular, as is required (for example) for any detailed analysis comparing latencies for single-sender and multi-sender protocols.

Communication complexity. Communication complexity refers to the number of bits transmitted by all correct parties. The Dolev-Reischuk bound states that the number of messages sent by correct parties (and so also the communication complexity) is $\Omega(f^2)$ in the worst case for Byzantine Broadcast/Agreement [37] (where f is the number of Byzantine parties). Intuitively, this is because every Byzantine party can request some message from every correct party. Communication complexity has been thought to be directly related to the throughput of the protocol, since sending more protocol metadata implies sending fewer transaction data parcels for the same total amount of data sent. Thus, in BFT protocols, we have seen works that focus on improving the communication complexity from exponential communication [38] to $O(n^4)$ [2] to $O(n^3)$ [4], [39], [40] to $O(n^2)$ [7], [8], [41], [42] to sub-quadratic [43], [44], [45].

Extension protocols, data availability oracles, pipelining. The line of work on extension protocols explicitly considers consensus protocols for ℓ-bit long messages where ℓ is a parameter. For a large ℓ, they obtain a communication complexity of $O(n\ell + \mathsf{poly}(n)\kappa)$ where κ is a security parameter [46], [47], [48], [49]. These protocols rely on erasure coding techniques to improve communication complexity, and thus improve throughput. Data availability oracles such as EigenDA and Tiramisu also use related ideas to separate data dissemination from consensus to improve the throughput of the protocol [50], [51], [52], [53], [54], [55], [56]. Finally, several state machine replication protocols have considered "pipelining", i.e., transmitting a block before committing the previous block it extends, as a means to improve throughput and latency [7], [40], [57], [58].

DAG-based protocols. It has often been observed that, in practice, low communication complexity does not necessarily imply high throughput. Narwhal and Tusk [59] showed that building on a DAG-based dissemination layer, where each party is responsible for transmitting transactions, allows for a much higher throughput. In fact, Narwhal is a system-level optimization where every party can utilize several *worker nodes* to transmit data and scale the throughput (albeit at the cost of higher underlying network bandwidth). The use of several worker machines implies using several CPUs and larger bandwidth. While larger bandwidth can be incorporated in our model, the use of several CPUs is not captured by our model. Due to the transmission of data and protocol messages by several parties at once, these works typically incur higher communication complexity. In addition to being a dissemination layer, several works rely on the underlying DAG structure to reach consensus [9], [10], [11], [12], [13], [14], [60], [61], [62], [63]. Early DAG-based protocols had high round-latency and so improving the round-latency to match that of non-DAG-based counterparts has been an active area of research.

Quorum-based vs. Nakamoto-style protocols. All of our analysis in this work has been designed with a focus on "quorum-based" protocols. However, there exist several "Nakamoto-style" works that have focused on improving latency and communication complexity [16], [64], [65], [66]. The methods developed in this work can also be applied to the analysis of Nakamoto-style protocols.

8. Conclusion and Future Work

Designing SMR protocols with low latency and high throughput has been a highly active area of research. This has been particularly true over the last decade, since the launch of Bitcoin [67] initiated worldwide interest in 'blockchains'. However, research in this area has been bottlenecked by existing analytical tools, which measure latency via round-complexity, relying on communication complexity to take some account of message sizes. The stark distinction between theory and practice became obvious to the community due to the release and adoption of DAG-based protocols. For the first time, these protocols challenged the narrative that low communication complexity implies the ability to deal with high throughputs. DAG-based protocols were seen to produce low latency in practice, even if theoretical metrics do not reflect this fact. In achieving these goals, the community introduced optimizations such as Narwhal [59].

In this paper, we take a first-principles approach to understanding and analyzing network-level performance. The key contribution of the paper is a new approach to analyzing latency and throughput — our latency measure considers the network delay, the need to send data parcels, and the complex interaction between different processors to actually deliver these messages. This approach combines communication complexity and traditional round-latency in a single metric, while providing bounds on what throughput can be obtained by a protocol.

One key result of our analysis is a comparison between latency for leader-based and DAG-based protocols. While today it is colloquially understood that DAG-based protocols outperform leader-based protocols in latency and throughput, our analysis suggests that there are trade-offs between the two approaches. At a high level, the key to obtaining low latency and high throughputs is (as far as possible) to use the full bandwidth of each processor at all times, so that no single party is a bottleneck. With leader-based protocols, one can use erasure codes to achieve this. With this approach, leader-based protocols, that rely on a fixed leader who pipelines data dissemination, have the ability to achieve low latency until a 'latency bottleneck' is reached for sufficiently high throughputs. The advantage of DAG-based protocols is that the latency bottleneck occurs at throughput which is higher by a constant factor. This is because erasure coding incurs sending a constant factor more data. There are other works, such as Autobahn [15] and BBCA-Chain [68], that use ideas from both of these approaches. Analyzing and comparing these works using our approach is an avenue for future work.

Another key result is a latency comparison between HotStuff and Tendermint. HotStuff reduced communication complexity to linear communication complexity per-view so that the protocol can be scaled to a large number of parties. This was achieved by sending data through the leader at the expense of increasing the number of rounds. On the other hand, Tendermint incurred a quadratic communication complexity. Our latency analysis shows that HotStuff does not outperform Tendermint in terms of latency or the latency bottleneck for any value of n. The leader indeed acts as a bottleneck, and the approach does not improve either latency or the maximum throughput obtained. We do, however, note that protocols may still potentially use HotStuff as a sub-protocol to achieve improved latency and throughput. For instance, the VABA protocol [69] uses n parallel instances of HotStuff, where every party acts as a leader for different instances of HotStuff and thus performs the same amount of work. It may be the case that the use of Hotstuff in this context allows for a higher latency bottleneck (high throughput).

While our analysis is a significant step forward, it makes several simplifying assumptions, leaving a number of avenues for future work. One large simplification is that our analysis only deals with the best case scenario: latency analysis is for the case that everyone is correct and the network is synchronous. Of course, processor or network failure can significantly impact latency, but the precise impact may not be obvious. For instance, if a leader fails in a single-sender protocol, it is clear that latency increases, but if we also expect the next leader to transmit double the amount of transaction data, this will have further knock-on effects impacting latency. In a DAG-based protocol, in contrast, data is still disseminated by honest processors when the leader is faulty. Similarly, while Δ may be an upper bound on network delays, shorter message delays will generally also be possible. For instance, the geographic distribution of processors will not generally be uniform and, even if it is, in practice there may be variance in message delays over time. As an example, compare two protocols requiring two different quorum sizes. In the presence of stragglers, a protocol that requires a 51% quorum may be faster than one that requires a 95% quorum. Also, the metric we compare is the worst case latency under the best case scenario. Considering other metrics such as average case or expected case or best case is an avenue for future work.

As discussed earlier, our analysis relies on a simplified network model:

(i) We abstract away low-level details of protocols like TCP, such as retransmissions, buffer sizes, and congestion control.

(ii) We assume full connectivity—every party can directly communicate with every other; extending this to peer-to-peer settings is left for future work.

(iii) We assume processors can send arbitrarily small parcels, though practical networks may enforce minimum payload sizes of tens or hundreds of bytes.

Importantly, our analysis assumes computation is free. In practice, operations such as erasure coding and digital signatures incur non-trivial costs that impact throughput and latency. These CPU cycles may also affect other components of the blockchain system, such as transaction execution.

References

[1] M. J. Fischer, N. A. Lynch, and M. S. Paterson, "Impossibility of distributed consensus with one faulty process," *Journal of the ACM*

(JACM), vol. 32, no. 2, pp. 374–382, 1985.

[2] C. Dwork, N. Lynch, and L. Stockmeyer, "Consensus in the presence of partial synchrony," vol. 35, no. 2. ACM New York, NY, USA, 1988, pp. 288–323.

[3] F. B. Schneider, "Implementing fault-tolerant services using the state machine approach: A tutorial," *ACM Computing Surveys (CSUR)*, vol. 22, no. 4, pp. 299–319, 1990.

[4] M. Castro, B. Liskov *et al.*, "Practical byzantine fault tolerance," in *OSDI*, ser. 99, no. 1999, 1999, pp. 173–186.

[5] E. Buchman, J. Kwon, and Z. Milosevic, "The latest gossip on BFT consensus," *CoRR*, vol. abs/1807.04938, 2018. [Online]. Available: http://arxiv.org/abs/1807.04938

[6] E. Buchman, "Tendermint: Byzantine fault tolerance in the age of blockchains," Ph.D. dissertation, University of Guelph, 2016.

[7] M. Yin, D. Malkhi, M. K. Reiter, G. G. Gueta, and I. Abraham, "Hotstuff: Bft consensus with linearity and responsiveness," in *ACM PODC'19*, 2019, pp. 347–356.

[8] D. Malkhi and K. Nayak, "Hotstuff-2: Optimal two-phase responsive bft," *Cryptology ePrint Archive*, 2023.

[9] I. Keidar, E. Kokoris-Kogias, O. Naor, and A. Spiegelman, "All you need is dag," in *ACM PODC'21*, 2021, pp. 165–175.

[10] A. Spiegelman, N. Giridharan, A. Sonnino, and L. Kokoris-Kogias, "Bullshark: The partially synchronous version," *arXiv preprint arXiv:2209.05633*, 2022.

[11] B. Arun, Z. Li, F. Suri-Payer, S. Das, and A. Spiegelman, "Shoal++: High throughput {DAG}{BFT} can be fast and robust!" in *USENIX NSDI '25*, 2025, pp. 813–826.

[12] N. Shrestha, R. Shrothrium, A. Kate, and K. Nayak, "Sailfish: Towards improving the latency of dag-based bft," in *2025 IEEE S&P*. IEEE, 2025, pp. 1928–1946.

[13] I. Keidar, O. Naor, O. Poupko, and E. Shapiro, "Cordial miners: Fast and efficient consensus for every eventuality," in *DISC'23*. Schloss-Dagstuhl-Leibniz Zentrum für Informatik, 2023.

[14] K. Babel, A. Chursin, G. Danezis, L. Kokoris-Kogias, and A. Sonnino, "Mysticeti: Low-latency dag consensus with fast commit path," *arXiv preprint arXiv:2310.14821*, 2023.

[15] N. Giridharan, F. Suri-Payer, I. Abraham, L. Alvisi, and N. Crooks, "Autobahn: Seamless high speed bft," in *ACM SOSP'24*, 2024, pp. 1–23.

[16] V. Bagaria, S. Kannan, D. Tse, G. Fanti, and P. Viswanath, "Prism: Deconstructing the blockchain to approach physical limits," in *ACM CCS'19*, 2019, pp. 585–602.

[17] J. Neu, S. Sridhar, L. Yang, D. Tse, and M. Alizadeh, "Longest chain consensus under bandwidth constraint," in *ACM AFT'22*, 2022, pp. 126–147.

[18] A. Buchwald, S. Buttolph, A. Lewis-Pye, P. O'Grady, and K. Sekniqi, "Frosty: Bringing strong liveness guarantees to the snow family of consensus protocols," *arXiv preprint arXiv:2404.14250*, 2024.

[19] N. Polyanskii, S. Mueller, and I. Vorobyev, "Starfish: A high throughput bft protocol on uncertified dag with linear amortized communication complexity," *Cryptology ePrint Archive*, 2025.

[20] V. Shoup, "Sing a song of simplex," *Cryptology ePrint Archive*, 2023.

[21] G. Bracha, "Asynchronous byzantine agreement protocols," *Information and Computation*, vol. 75, no. 2, pp. 130–143, 1987.

[22] Ethereum, "Proposer builder separation," https://ethereum.org/en/roadmap/pbs/.

[23] A. Foundation, "Decentralized timeboost specification," https://research.arbitrum.io/t/decentralized-timeboost-specification/9676.

[24] A. Author(s), "Single-sender beb implementation," https://anonymous.4open.science/r/pipes-082D.

[25] ——, "Multi-sender beb implementation," https://anonymous.4open.science/r/pipes-FFBB.

[26] ——, "Sailfish implementation," https://anonymous.4open.science/r/pipes-744B.

[27] ——, "Disperedsimpex implementation," https://anonymous.4open.science/r/dsimplex-D814.

[28] [n.d.], "Google cloud platform - general-purpose machine family for compute engine." online; accessed 06-April-2025. [Online]. Available: https://cloud.google.com/compute/docs/general-purpose-machines#e2_machine_types

[29] ——, "Linux traffic control," online; accessed 06-Nov-2025. [Online]. Available: https://linux.die.net/man/8/tc

[30] L. Foundation, "Tc-tbf linux manual page," https://man7.org/linux/man-pages/man8/tc-tbf.8.html.

[31] I. Abraham, K. Nayak, L. Ren, and Z. Xiang, "Good-case latency of byzantine broadcast: A complete categorization," in *ACM PODC'21*, 2021, pp. 331–341.

[32] M. J. Fischer and N. A. Lynch, "A lower bound for the time to assure interactive consistency," *Information processing letters*, vol. 14, no. 4, pp. 183–186, 1982.

[33] I. Abraham, S. Devadas, D. Dolev, K. Nayak, and L. Ren, "Synchronous byzantine agreement with expected o (1) rounds, expected communication, and optimal resilience," in *FC'19*. Springer, 2019, pp. 320–334.

[34] C.-D. Sandro, M. Fitzi, A. Kiayias, G. Panagiotakos, and A. Russell, "High-throughput blockchain consensus under realistic network assumptions," in *https://iohk.io/en/research/library/papers/high-throughput-blockchain-consensus-under-realistic-network-assumptions/*, 2024.

[35] L. Kiffer, J. Neu, S. Sridhar, A. Zohar, and D. Tse, "Nakamoto consensus under bounded processing capacity," in *ACM CCS'24*, 2024, pp. 363–377.

[36] J. Garay, A. Kiayias, and N. Leonardos, "The bitcoin backbone protocol: Analysis and applications," *Journal of the ACM*, vol. 71, no. 4, pp. 1–49, 2024.

[37] D. Dolev and R. Reischuk, "Bounds on information exchange for byzantine agreement," *Journal of the ACM (JACM)*, vol. 32, no. 1, pp. 191–204, 1985.

[38] L. Lamport, R. Shostak, and M. Pease, "The byzantine generals problem," in *Concurrency: the works of leslie lamport*, 2019, pp. 203–226.

[39] E. Buchman, J. Kwon, and Z. Milosevic, "The latest gossip on bft consensus," *arXiv preprint arXiv:1807.04938*, 2018.

[40] V. Buterin and V. Griffith, "Casper the friendly finality gadget," *arXiv preprint arXiv:1710.09437*, 2017.

[41] A. Lewis-Pye, "Quadratic worst-case message complexity for state machine replication in the partial synchrony model," *arXiv preprint arXiv:2201.01107*, 2022.

[42] P. Civit, M. A. Dzulfikar, S. Gilbert, V. Gramoli, R. Guerraoui, J. Komatovic, and M. J. Ribeiro Vidigueira, "Byzantine consensus is θ (nˆ 2): The dolev-reischuk bound is tight even in partial synchrony!" *DISC'22*, no. 11, pp. 1–11, 2022.

[43] I. Abraham, T. H. Chan, D. Dolev, K. Nayak, R. Pass, L. Ren, and E. Shi, "Communication complexity of byzantine agreement, revisited," in *ACM PODC'19*, 2019, pp. 317–326.

[44] V. King, J. Saia, V. Sanwalani, and E. Vee, "Scalable leader election," in *SODA'06*, 2006, pp. 990–999.

[45] Y. Gilad, R. Hemo, S. Micali, G. Vlachos, and N. Zeldovich, "Algorand: Scaling byzantine agreements for cryptocurrencies," in *ACM SOSP'17*, 2017, pp. 51–68.

[46] A. Bhangale, C.-D. Liu-Zhang, J. Loss, and K. Nayak, "Efficient adaptively-secure byzantine agreement for long messages," in *ASIACRYPT'22*. Springer, 2022, pp. 504–525.

[47] K. Nayak, L. Ren, E. Shi, N. H. Vaidya, and Z. Xiang, "Improved extension protocols for byzantine broadcast and agreement," in *DISC'20*, 2020.

[48] C. Ganesh and A. Patra, "Optimal extension protocols for byzantine broadcast and agreement," *Distrib. Comput.*, vol. 34, no. 1, p. 59–77, Feb. 2021. [Online]. Available: https://doi.org/10.1007/s00446-020-00384-1

[49] J. A. Garay, R. Gennaro, C. Jutla, and T. Rabin, "Secure distributed storage and retrieval," *Theor. Comput. Sci.*, vol. 243, no. 1–2, p. 363–389, jul 2000. [Online]. Available: https://doi.org/10.1016/S0304-3975(98)00263-1

[50] EigenLabs, "Intro to eigenda: Hyperscale data availability for rollups," 2023, accessed on March 20, 2024. [Online]. Available: https://www.blog.eigenlayer.xyz/intro-to-eigenda-hyperscale-data-availability-for-rollups/

[51] Ethereum, "Data availability — ethereum.org," 2024, accessed on March 20, 2024. [Online]. Available: https://ethereum.org/en/developers/docs/data-availability/

[52] E. Systems, "Designing the espresso sequencer: Combining hotshot consensus with tiramisu da," 2023, accessed on March 20, 2024. [Online]. Available: https://hackmd.io/@EspressoSystems/HotShot-and-Tiramisu

[53] L. Yang, S. J. Park, M. Alizadeh, S. Kannan, and D. Tse, "{DispersedLedger}:{High-Throughput} byzantine consensus on variable bandwidth networks," in *USENIX NSDI '22*, 2022, pp. 493–512.

[54] K. Nazirkhanova, J. Neu, and D. Tse, "Information dispersal with provable retrievability for rollups," in *AFT'22*, 2022, pp. 180–197.

[55] S. Cohen, G. Goren, L. Kokoris-Kogias, A. Sonnino, and A. Spiegelman, "Proof of availability and retrieval in a modular blockchain architecture," in *FC'23*. Springer, 2023, pp. 36–53.

[56] A. Skidanov and I. Polosukhin, "Nightshade: Near protocol sharding design," *URL: https://nearprotocol. com/downloads/Nightshade. pdf*, vol. 39, 2019.

[57] I. Abraham, D. Malkhi, K. Nayak, L. Ren, and M. Yin, "Sync hotstuff: Simple and practical synchronous state machine replication," in *IEEE S&P'20*. IEEE, 2020, pp. 106–118.

[58] N. Shrestha, I. Abraham, L. Ren, and K. Nayak, "On the optimality of optimistic responsiveness," in *ACM CCS'20*, 2020, pp. 839–857.

[59] G. Danezis, L. Kokoris-Kogias, A. Sonnino, and A. Spiegelman, "Narwhal and tusk: a dag-based mempool and efficient bft consensus," in *Eurosys'22*, 2022, pp. 34–50.

[60] A. Gkagol, D. Lesniak, D. Straszak, and M. Swietek, "Aleph: Efficient atomic broadcast in asynchronous networks with byzantine nodes," in *AFT'19*, 2019, pp. 214–228.

[61] X. Dai, G. Wang, J. Xiao, Z. Guo, R. Hao, X. Xie, and H. Jin, "Lightdag: A low-latency dag-based bft consensus through lightweight broadcast," *Cryptology ePrint Archive*, 2024.

[62] G. Danezis and D. Hrycyszyn, "Blockmania: from block dags to consensus," *arXiv preprint arXiv:1809.01620*, 2018.

[63] A. Spiegelman, B. Aurn, R. Gelashvili, and Z. Li, "Shoal: Improving dag-bft latency and robustness," in *FC'24*, 2024.

[64] C. Li, P. Li, D. Zhou, Z. Yang, M. Wu, G. Yang, W. Xu, F. Long, and A. C.-C. Yao, "A decentralized blockchain with high throughput and fast confirmation," in *USENIX ATC'20*, 2020, pp. 515–528.

[65] H. Yu, I. Nikolić, R. Hou, and P. Saxena, "Ohie: Blockchain scaling made simple," in *IEEE S&P'20*, 2020, pp. 90–105.

[66] L. Yang, Y. Gilad, and M. Alizadeh, "Coded transaction broadcasting for high-throughput blockchains," 2022. [Online]. Available: https://arxiv.org/abs/2205.01797

[67] S. Nakamoto, "Bitcoin: A peer-to-peer electronic cash system," 2008.

[68] D. Malkhi, C. Stathakopoulou, and M. Yin, "Bbca-chain: One-message, low latency bft consensus on a dag," in *International Conference on Financial Cryptography and Data Security*, 2024.

[69] I. Abraham, D. Malkhi, and A. Spiegelman, "Asymptotically optimal validated asynchronous byzantine agreement," in *ACM PODC'19*, 2019, pp. 337–346.

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

[71] M. O. Rabin, "Efficient dispersal of information for security, load balancing, and fault tolerance," *Journal of the ACM (JACM)*, vol. 36, no. 2, pp. 335–348, 1989.

[72] D. Boneh, B. Lynn, and H. Shacham, "Short signatures from the weil pairing," in *International conference on the theory and application of cryptology and information security*. Springer, 2001, pp. 514–532.

[73] V. Shoup, "Practical threshold signatures," in *Advances in Cryptology—EUROCRYPT 2000: International Conference on the Theory and Application of Cryptographic Techniques Bruges, Belgium, May 14–18, 2000 Proceedings 19*. Springer, 2000, pp. 207–220.

Appendix A.
BeB for the multi-sender setting

The protocol is specified in Figures 9 and 10. Next, we prove Claim 2.

Initialize
$\mathsf{block\text{-}txns} = \{\}$
$\mathsf{sndbuf} = \{\}$
$\forall j\ \mathsf{recv\text{-}block}_j = \{\}$

Collect txns
Collect *txns*, which is all data parcels received by the client processor but not yet added to sndbuf
$\mathsf{block\text{-}txns} = \mathsf{block\text{-}txns} \cup \mathit{txns}$
$\mathsf{sndbuf} = \mathsf{sndbuf} || \mathit{txns}$

Transfer to upload
Dequeue the first S/n parcels in sndbuf and add each to the upload buffer n times, once for each recipient

Receive blocks
upon receiving block data parcels m from p_j:
$\mathsf{recv\text{-}block}_j = \mathsf{recv\text{-}block}_j || m$
If $\mathsf{fullblock}(\mathsf{recv\text{-}block}_j) = \mathsf{true}$:
Deliver block received from p_j
$\mathsf{recv\text{-}block}_j = \{\}$

Figure 9: Some procedures used in Figure 10 (and in later sections)

The proof of Claim 2. The proof is essentially the same as the proofs of Claim 1. The only difference here is that every processor is performing dissemination, and is correspondingly also receiving data from every other processor. Since no processor sends more than S/n block data parcels in each time slot, no processor receives more than $n \times S/n$ parcels in a time slot. This means that download buffers are not a bottleneck. □

The approach in which all processors transmit data is commonly used in DAG-based protocols where each processor's block for a given 'layer' references $\Theta(n)$ blocks from

The following instructions are for p_i.
at time slot 0 **do**
  Initialize ▷ As specified in Figure 9
at each time slot t **do**
  Collect txns ▷ As specified in Figure 9
  Transfer to upload ▷ As specified in Figure 9

  if $|\mathsf{block\text{-}txns}| = \frac{\alpha}{1-\alpha}M$ **then** ▷ M denotes the total size of metadata
    ▷ $\alpha = D/S$
    Set $\mathrm{md}^{\mathrm{de}}$ = *dependent* metadata for the current block
    Set $\mathrm{md}^{\mathrm{in}}$ = *independent* metadata for the next block
    $\mathsf{sndbuf} = \mathsf{sndbuf}||\mathrm{md}^{\mathrm{de}}||\mathrm{md}^{\mathrm{in}}$
    $\mathsf{block\text{-}txns} = \{\}$
  Receive blocks ▷ As specified in Figure 9

Figure 10: Best-effort Broadcast by each processor

the previous layer (e.g. [9]). However, the need to receive blocks from layer d before producing blocks for layer $d+1$ produces certain complications in the latency analysis. For an analysis of these considerations, see Appendix D.

Appendix B. Clearing times

The proof of Claims 1 and 2 introduce two notions that will be useful in later sections:

(a) *The extended clearing time*. Consider a timeslot t at which the protocol of Figure 4 places metadata of size M in sendbuffer (which is empty prior to this addition). If nM is small compared to S then this data will be cleared quickly from sendbuffer. If nM is large compared to S, however, then a backlog of transactions will begin to build up in sendbuffer starting at t. Further transactions arrive as we work to clear this backlog, adding to the time required to clear the backlog and reach a state in which all received transactions have been removed from sendbuffer. According to the analysis given in the proof, the backlog will finally be cleared by time $t + \frac{Mn}{(1-\alpha)S}$. While the analysis in the proof specifically considered metadata of size M, exactly the same analysis holds for *any* set of data of arbitrary size X (say). We may therefore consider the *extended clearing time* for a set of data of size X to be:

$$EC(X) := \frac{Xn}{(1-\alpha)S}$$

(b) *The clearing time*. In the proof of Claim 1 we also had to consider the time $M^{\mathrm{de}}n/S$, which is the time it takes to clear the dependent metadata from sendbuffer. More generally, we can consider the *clearing time* for a set of data of size X to be:

$$C(X) := \frac{Xn}{S}$$

Expressed using these terms, latency for the protocol of Figure 4 is:

$$EC(M) + C(M^{\mathrm{de}}) + \Delta.$$

Appendix C. Multi-sender BeB with a layered DAG

So far, our analysis for multi-sender BeB applies to a scenario in which each processor can choose n previous blocks to point to at the beginning of the process of forming a new block. However, many DAG-based protocols build a more structured DAG consisting of 'layers': each block in layer $d+1$ must reference (via hash pointers in the metadata) $\Theta(n)$ blocks from layer d. Analyzing latency for protocols of this form introduces certain complexities:

(a) Correct processors may not be perfectly 'in sync'. In particular, processors may not begin building their blocks for a given layer at the same time as each other.
(b) Even if each correct processor begins construction for layer $d+1$ at exactly the same time, processors must wait to receive blocks in layer d before they can determine the hash pointers to be included in the metadata for their block in layer $d+1$.

In this section, we subdue complexity (a) above, i.e. for the sake of simplicity, we consider a scenario in which correct processors begin building their blocks for each layer simultaneously. In this simplified setting, we consider the impact of complexity (b) on latency. Setting $\alpha = D/S$, this leads to three regimes of interest:

(i) **Regime A**: $EC(M^{\mathrm{de}}) \geq C(M^{\mathrm{de}}) + \Delta$.
(ii) **Regime B**: Regime A does not hold, but $EC(M) \geq C(M) + \Delta$.
(iii) **Regime C**: Neither of the above.

We consider these three regimes separately in what follows.

Regime A. In this case, suppose that all processors (perfectly synchronized) finish production of layer d at t by adding M^{de} to their buffer, but have not yet received full blocks for layer d from other processors. Each processor will then receive the layer d blocks of other processors by $t + C(M^{\mathrm{de}}) + \Delta$, fully using their bandwidth all the while, and so can add M^{in} (the pointers to blocks from the previous layer) at this time. Figure 11 shows the resulting protocol. Since the time to clear the block metadata and all incoming transactions from sendbuffer is unchanged depending on whether we add M^{in} to sendbuffer at t or $t+C(M^{\mathrm{de}})+\Delta$, the latency analysis is then essentially the same as for Section 3.2. As before, the latency is:

$$EC(M) + C(M^{\mathrm{de}}) + \Delta = \frac{(M + (1-\alpha)M^{\mathrm{de}})n}{(1-\alpha)S} + \Delta.$$

Regime B. In this case, we can ensure that processors use their entire bandwidth at each time slot by adding all metadata at the end of the block. The resulting protocol is shown in Figure 12. This gives latency of:

$$EC(M) + C(M) + \Delta = \frac{(2-\alpha)Mn}{(1-\alpha)S} + \Delta.$$

The following instructions are for p_i.
at time slot 0 **do**
  Initialize ▷ As specified in Figure 9
  Set $d = 1$
at each time slot t **do**:
  Collect txns ▷ As specified in Figure 9
  Transfer to upload ▷ As specified in Figure 9
  If $d > 1$ and received layer $d-1 = \mathsf{true}$
    Set $\mathsf{md}^{\mathsf{in}}$ = independent metadata for the current block
    Set $\mathsf{sndbuf} = \mathsf{sndbuf}||\mathsf{md}^{\mathsf{in}}$
  If $|\mathsf{block\text{-}txns}| \geq \frac{\alpha}{1-\alpha}M$ ▷ $\alpha = D/S$
    Set $\mathsf{md}^{\mathsf{de}}$ = dependent metadata for the current block
    $\mathsf{sndbuf} = \mathsf{sndbuf}||\mathsf{md}^{\mathsf{de}}$
    $\mathsf{block\text{-}txns} = \{\}$
    $d := d + 1$
  Receive blocks ▷ As specified in Figure 9

Figure 11: BeB for a layered DAG in Regime A

The following instructions are for p_i.
at time slot 0 **do**
  Initialize
  Set $d = 1$
at each time slot t **do**:
  Collect txns
  Transfer to upload
  If sndbuf is free of transactions **and** ($d = 1$ **or** received layer $d-1 = \mathsf{true}$):
    Set $\mathsf{md}^{\mathsf{in}}$ = independent metadata for the current block
    Set $\mathsf{md}^{\mathsf{de}}$ = dependent metadata for the current block
    $\mathsf{sndbuf} = \mathsf{sndbuf}||\mathsf{md}^{\mathsf{in}}||\mathsf{md}^{\mathsf{de}}$
    $d := d + 1$
  Receive blocks

Figure 12: BeB for a layered DAG in Regimes B or C

Regime C. In this case, the protocol of Figure 12 still applies, but now the latency analysis is different, because the delay Δ dominates the wait to receive blocks from the previous layer. For the same protocol, latency is now:

$$2(C(M) + \Delta) = \frac{2Mn}{S} + 2\Delta.$$

This can be explained as follows. Let us say t is the time at which metadata for layer $d-1$ was added to the sndbuf by p_i. So, any transaction arriving at p_i after time t will be sent as a part of layer d. The metadata sent by p_i at time t will be received by all parties at time $t + C(M) + \Delta$; similarly, party p_i will receive layer $d-1$ metadata from all parties at time $t + C(M) + \Delta$. Party p_i keeps sending transactions for layer d starting at time t. Once it has received metadata for layer $d-1$ (at time $t + C(M) + \Delta$), it will start adding metadata for layer d in the sndbuf. This will arrive at all parties at $t + 2C(M) + 2\Delta$.

One can view the first $C(M) + \Delta$ term from layer $d-1$ as being the queueing delay for layer d. The latter $C(M) + \Delta$ constitutes the time for completing the layer d block. Observe that, the transactions corresponding to layer d are pipelined; they are sent in the intervening time before layer $d-1$ blocks are entirely received. Moreover, the value of $C(M) + \Delta$ determines the block size.

Appendix D. Lower bounds for BeB

Intuitively, it may seem clear that the protocol of Figure 4 is the best one can possibly do if one wishes to minimize latency for a single-sender protocol, and if one is restricted to using a protocol in which the leader must send *all* data to *all* processors without using methods like erasure coding or a gossip network. However, the question remains: is there a formal sense in which one can establish that the latency of the protocol is optimal? To answer this question we will need some new techniques. First, though, we need to formalise what we mean by 'protocols that don't use techniques like erasure coding or a gossip network'.

Simple-broadcast protocols. We say that a protocol is a *simple-broadcast* protocol if both:

1) Correct processors send the same blocks to all processors, i.e., if p_i is correct and sends a block b to p_j then p_i must send b to all processors, and;
2) Each correct processor p_i *delivers* a transaction initially received by the client processor of p_j when p_i receives a full block b from p_j containing that transaction amongst its transaction parcels,[13] i.e., p_i must receive every block data parcel m of b from p_j before transactions in the block are delivered.[14]

The intention of this definition is to exclude the use of techniques such as erasure coding or a gossip network for improving latency. The definition does not rule out standard approaches to Best-effort Broadcast, such as the leader simply waiting until a full block is formed and then sending it to all processors. However, the protocol of Figure 4 is rather specific in the way it separately treats independent and dependent metadata, and has lower latency than such standard approaches.

The following claim establishes that the protocol of Figure 4 is exactly optimal amongst simple-broadcast protocols for the single-sender setting.

Claim 4. *Consider the single-sender setting, suppose all processors are correct, and let M and M^{de} be as defined*

13. So that p_i can determine when to deliver a transaction, one may suppose that transactions contain within their data the identity of the client processor to which they are sent.

14. In particular, it does not suffice that p_i receives information (from p_j or other processors) which suffices to *recover* the block: p_i must actually receive every data parcel in b from p_j before delivering the transactions in b.

previously. If $D = \alpha S/n$ for $\alpha \in (0,1)$, then latency for any simple-broadcast protocol is at least:

$$f(M, M^{de}, D, S, n, \Delta) := \frac{(M + (1-\alpha)M^{de})n}{(1-\alpha)S} + \Delta.$$

Proof. By the definition of a simple-broadcast protocol, each block sent by p_ℓ must be sent to all processors. So, for each block b sent by p_ℓ, we can consider the first timeslot, $t_1(b)$ say, by which it holds, for every data parcel m of b and every processor p_i, that p_ℓ has removed a copy of m addressed to p_i from its upload buffer (meaning that b will be received by all processors by $t_1(b) + \Delta$).

We can assume that each transaction received by the client processor of p_ℓ is included in precisely one block sent by the leader. To see this, note that, if some transaction is not included in any block, then latency is unbounded. If a transaction `tr` is included in two blocks, b_1 and b_2 say, then, without loss of generality, suppose that $t_1(b_2) \geq t_1(b_1)$. In this case, one can remove `tr` from b_2 without increasing latency. We can also assume that each block sent by p_ℓ includes at least one transaction, since one can eliminate the sending of empty blocks without increasing latency.

The basic idea. For each block b sent by p_ℓ, let $t_0(b)$ be the first time slot at which p_ℓ receives a transaction at its client processor that is included in b. We define the *block latency for* b to be $\ell(b) := t_1(b) + \Delta - t_0(b)$. Note that $\ell(b)$ lower bounds latency, since at least one transaction in b is received at client processor of p_ℓ at t_0, and $t_1(b) + \Delta$ is the first time slot at which transactions in b are delivered by all correct processors. The basic idea behind the remainder of the proof is to use an averaging argument to show that *mean* block latency is at least $f(M, M^{\text{de}}, D, S, n, \Delta)$, so that the latter value must lower bound $\ell(b)$ for at least one b. (In fact, mean block latency might not be defined, i.e., might not come to a limit as the number of blocks considered tends to infinity, so we have to be careful in formalising this idea, but the above indicates the spirit of the argument that follows.)

Accountancy for time slots in $[t_0(b), t_1(b)]$. To lower bound $t_1(b) - t_0(b)$, we consider which data parcels are removed from the upload buffer of p_ℓ at each time slot in the interval $[t_0(b), t_1(b)]$. Let $M^{\text{in}}_{-}(b)$ be the number of addressed independent metadata parcels for b which are removed from the upload buffer at time slots strictly *before* $t_0(b)$, and let $M^{\text{in}}_{+}(b)$ be the number of addressed independent metadata parcels for b which are removed from the upload buffer at time slots at or *after* $t_0(b)$. If $B(b)$ transaction parcels are included in b then:

(i) Removing addressed transactions in b from the upload buffer accounts for at least $B(b)n/S$ timeslots in $[t_0(b), t_1(b)]$;

(ii) Removing addressed dependent metadata parcels for b accounts for at least $M^{\text{de}}n/S$ timeslots in $[t_0(b), t_1(b)]$, and;

(iii) Removing addressed independent metadata parcels for b accounts for a further $M^{\text{in}}_{+}(b)/S$ timeslots in $[t_0(b), t_1(b)]$.

However, we must also consider that there may be time slots in the interval $[t_0(b), t_1(b)]$ at which the leader removes data parcels corresponding to blocks *other than* b from the upload buffer: suppose that removing data parcels corresponding to blocks other than b from the upload buffer accounts for $x(b)$ time slots in $[t_0(b), t_1(b)]$.

Let the blocks sent by the leader be $b_1, b_2, \ldots$, ordered by $t_1(b)$ and with ties broken by least hash. For each $r \in \mathbb{N}_{\geq 1}$, set $\mathbb{E}_r[\ell(b)] = \sum_{i=1}^{r} \ell(b_r)/r$. We also extend this notation in the obvious way to other terms, such as $\mathbb{E}_r[x(b)]$. By linearity of expectation and the observations above it follows that:

$$\mathbb{E}_r[\ell(b)] \geq \frac{n\,\mathbb{E}_r[B(b)]}{S} + \frac{nM^{\text{de}}}{S} + \frac{\mathbb{E}_r[M^{\text{in}}_{+}(b)]}{S} + \mathbb{E}_r[x(b)] + \Delta. \tag{9}$$

The remainder of the proof consists of two steps:

1) We lower bound $\mathbb{E}_r[x(b)]$ for all sufficiently large r.
2) We lower bound $\mathbb{E}_r[B(b)]$ for all sufficiently large r.

Combining these bounds with (9) suffices to establish the claim.

Step 1. We aim to bound $\mathbb{E}_r[x(b)]$ for sufficiently large r. Since the dependent metadata for any block b cannot be determined until after all the transactions for the block have been received, the time slots at which the leader removes the dependent metadata for b from its upload buffer contribute at least nM^{de}/S to the $x(b')$ values of other blocks.[15] Similarly, the time slots prior to $t_0(b)$ at which the leader removes the $M^{\text{in}}_{-}(b)$ addressed independent metadata parcels for b from the upload buffer contribute $M^{\text{in}}_{-}(b)/S$ to the $x(b')$ values of other blocks. Let us write $b_i \rightharpoonup x(b_j)$ if metadata for b_i is removed from the upload buffer of p_ℓ in the interval $[t_0(b_j), t_1(b_j)]$, i.e., the removal of metadata for b_i from the upload buffer contributes positively to $x(b_j)$. Note that:

$(*)$ If latency is bounded, there must exist r^* such that it is never the case $b_i \rightharpoonup x(b_{i+r})$ for $r \geq r^*$.

To argue that $(*)$ holds, note that at most S blocks b' can share the same value $t_1(b')$. So, if $b_i \rightharpoonup x(b_{i+r})$, then $\ell(b_{i+r}) \geq r/S$.

Now consider the blocks $b_1, \ldots, b_r$. The idea is now to show that, if r is large, then almost all the blocks b_j in this sequence (all but at most the last r^*) contribute at least $nM^{\text{de}}/S + M^{\text{in}}_{-}(b_j)/S$ to $\sum_{i=1}^{r} x(b_i)$. This allows us to lower bound $\mathbb{E}_r[x(b)]$. For each $j < r - r^*$, the time slots at which the leader removes the dependent metadata for b_j from its upload buffer must contribute at least nM^{de}/S to $\sum_{i=1}^{r} x(b_i)$. Similarly, for $j < r - r^*$, the time slots prior to $t_0(b_j)$ at which the leader removes the $M^{\text{in}}_{-}(b_j)$ addressed

15. We allow a rounding error of one time slot here (which also applies in the proof of Claim 1). In the very first time slot at which p_ℓ removes dependent metadata for b from its upload buffer, it may be that no transactions arrive that are included in blocks other than b. However, for all subsequent time slots at which p_ℓ removes dependent metadata for b from its upload buffer, transactions will arrive at the client processor of p_ℓ and these transactions must be included in blocks other than b, because the dependent metadata for b has already been determined.

independent metadata parcels for b_j from the upload buffer contribute $M_-^{\text{in}}(b_j)/S$ to $\sum_{i=1}^{r} x(b_i)$. It follows that, for each $\epsilon > 0$, the following holds for all sufficiently large r:

$$\mathbb{E}_r[x(b)] \geq (1-\epsilon)\left(nM^{\text{de}}/S + \mathbb{E}_r[M_-^{\text{in}}(b)]/S\right). \quad (10)$$

Putting equations (9) and (10) together, and since $\mathbb{E}[M_-^{\text{in}}] + \mathbb{E}[M_+^{\text{in}}] = \mathbb{E}[M_-^{\text{in}} + M_+^{\text{in}}] = nM^{\text{in}}$, we conclude that for each $\epsilon > 0$, it holds for all sufficiently large r that:

$$\mathbb{E}_r[\ell(b)] \geq n\,\mathbb{E}_r[B(b)]/S + nM^{\text{de}}/S + (1-\epsilon)Mn/S + \Delta.$$

If latency is bounded, so that the $\liminf$ values below are defined, this means that:

$$\liminf_r \mathbb{E}_r[\ell(b)] \geq \frac{n \liminf_r \mathbb{E}_r[B(b)]}{S} + \frac{nM^{\text{de}}}{S} + \frac{Mn}{S} + \Delta. \quad (11)$$

Step 2. To lower bound $\liminf_r \mathbb{E}_r[B(b)]$, note that it takes time at least $t(r) := n\,\mathbb{E}_r(B)r/S + Mnr/S$ to remove blocks $b_1, \ldots, b_r$ from the upload buffer. Since $D \cdot t(r)$ many transactions arrive at the client processor of the leader in this time, for latency to be bounded we require that, for each $\epsilon > 0$, it holds for all sufficiently large r that:

$$r\,\mathbb{E}_r[B(b)] \geq (1-\epsilon)\left(D(n\,\mathbb{E}_r(B)r/S + Mnr/S)\right). \quad (12)$$

To see that (12) must hold, note that there must exist an upper bound, B^* say, on the number of transactions included in a block, if latency is to be bounded. For each r, let $y(r)$ be the set of transactions that have been received at the client processor of p_ℓ by $t(r)$, but which are not included in $b_1, \ldots, b_r$. The failure of (12) means that $|y(r)|$ is unbounded, i.e., can be arbitrarily large for large r. The transactions in $y(r)$ must be divided between at least $|y(r)|/B^*$ many blocks. Since at most S blocks b' can share the same value $t_1(b')$, some transaction in $y(r)$ has latency at least $|y(r)|/SB^*$.

From (12), it follows that:

$$\liminf_r \mathbb{E}_r[B(b)] \geq \frac{DMn}{S - nD}. \quad (13)$$

Putting steps 1 and 2 together. Putting (13) together with (11), it follows that:

$$\liminf_r \mathbb{E}_r[\ell(b)] \geq \frac{(M + (1-\alpha)M^{\text{de}})n}{(1-\alpha)S} + \Delta,$$

so that latency is also bounded by the r.h.s. of this inequality, as claimed. □

An almost identical proof suffices to establish the corresponding claim for the protocol of Figure 10.

Appendix E. Consistent Broadcast (single-sender)

The protocol is shown in Figure 13.

In analysing the latency, we make the following simplifications. We suppose that *votes*, i.e., messages of the form $(\text{VOTE}, d, H(b))$, are of a fixed size λ, that metadata is of size M, and we analyse latency in the case that all processors are correct.

There is a designated processor p_ℓ. The following instructions are for p_i.
at time slot 0 do:
  Initialize
  Set $d = 1$, `send` $=$ `true`, `vote` $=$ `true`

at time slot t **if** `send` $=$ `true` **and** $p_i = p_\ell$:
  Collect txns
  If $|$`block-txns`$| = B$ **do**:
    ▷ B as specified in Appendix E
    Set md = metadata for the current block
    Set `sndbuf` $=$ `sndbuf`$||$md
    ▷ d included in metadata
    Set `send` $=$ `false`

at every time slot t **do**:
  If `vote` $=$ `true` **and** p_i has received a first block b of depth d from p_ℓ **do**:
    Set `sndbuf` $=$ `sndbuf`$||(\text{VOTE}, d, H(b))$
    Set `vote` $=$ `false`
  If p_i has received $(\text{VOTE}, d, H(b'))$ from $n - f$ distinct processors (for some fixed b')
  and if p_i has also received b' **do**:
    Deliver b'
    Set `send` $=$ `true`, `vote` $=$ `true`, $d = d + 1$
  Transfer to upload ▷ As specified in Figure 9

Figure 13: CB by a designated sender

Claim 5. *Suppose all processors are correct. If $S > Dn$ and B is set to minimise latency, then latency for the protocol of Figure 13 is:*

$$\left(\frac{2(M+\lambda)n}{S} + 4\Delta\right)\left(1 + \frac{1}{2((S/Dn) - 1)}\right)$$

Proof. Let the clearing time C be as defined in Appendix B. Since the time between each block delivery is:

$$C(B + M) + C(\lambda) + 2\Delta,$$

and since D transactions arrive at the client processor of p_ℓ at each timeslot, for latency to be bounded we require:

$$B \geq \frac{D((B + M + \lambda)n + 2\Delta S)}{S}.$$

This means B is greater than or equal to:

$$\frac{D((M + \lambda)n + 2\Delta S)}{S - Dn}. \quad (14)$$

So, set B equal to this value.

Consider next what happens between time slot 0 and the first timeslot, t_1 say, at which p_ℓ adds metadata to `sndbuf`. At each timeslot in this interval, less than S/n transactions arrive at the client processor of p_ℓ (since $S > Dn$), and are immediately added to `sndbuf` (via the instruction "Collect

$txns$"). Each is then added to p_ℓ's upload buffer n times (once for each recipient, via the instruction "Transfer to upload"), with the upload buffer emptied of addressed transactions by the end of the timeslot. It follows that, by the end of time slot t_1, sndbuf does not contain any transactions and that the leader's upload buffer does not contain any addressed transactions.

Now suppose that, at some timeslot t, the leader completes a block, i.e. while its local value $\texttt{send} = \texttt{true}$, it adds the metadata for some block to its sndbuf. Suppose (inductively) that, by the end of timeslot t, its sndbuf does not contain any transactions and that its upload buffer does not contain any addressed transactions (i.e. all such parcels added to these buffers have been removed). All correct processors will then deliver the next block at time:

$$t' := t + C(M) + C(\lambda) + 2\Delta = t + B(S - Dn)/DS.$$

In the interval $(t, t']$, the leader p_ℓ will therefore receive $B(S - Dn)/S$ many transaction parcels, which will be immediately added to sndbuf at t'. To complete the block, the leader requires a further

$$B - B(S - Dn)/S = nBD/S$$

transaction parcels, which will arrive in time nB/S, i.e. $C(B)$, which is exactly the time that p_ℓ requires to clear sndbuf of all transactions. It therefore holds that when the leader completes the next block at $t'' := t' + nB/S$, its sndbuf does not contain any transactions and that its upload buffer does not contain any addressed transactions.

To calculate the latency, we have to consider the time from when transactions start building up at t until the next block (which the leader starts sending at t') is delivered by all correct processors. This is:

$$\frac{(B + 2(M + \lambda))n}{S} + 4\Delta.$$

Substituting in the value for B in (15) gives the latency as claimed. □

Appendix F. Consistent Broadcast (multi-sender)

The protocol is specified by Figures 14 and 15. Figure 14 defines the $\oplus$ function, which is used to evenly spread the sending of data parcels, and we presume that addressed data parcels contain information that allows processors to recover each m_i from $\oplus_I m_i$

Suppose $S > D$. We assume the reader is familiar with standard CB protocols in what follows and analyze only the resulting latency, rather than re-verifying correctness. Consider an arbitrary processor p_i. Let M be size of block metadata, let B be the size of the transaction data in each block, and let λ be the size of each vote. Recall that we suppose that processors are synchronized. For each depth d_i, p_i sends a block to all other processors, but then must send votes for n different blocks of that depth, with each

Suppose $I \subseteq \{1, \ldots, n\}$ and that, for each $i \in I$, m_i is a sequence of data parcels $m_{i,1}, \ldots, m_{i,x_i}$ (where $m_{i,x}$ is undefined for $x > x_i$). Then $\oplus_I m_i$ is the sequence of data parcels formed as follows:

Let $I = \{i_1, \ldots i_k\}$, where each $i_{k'} < i_{k'+1}$ for $k' \in [1, k)$.
Set $x^* = \max\{x_i : i \in I\}$
Set m to be the empty sequence.
For $x = 1$ to x^* do:
  For $j = 1$ to k do:
    If $m_{i_j,x}$ is defined, set $m := m||m_{i_j,x}$
Output m

Figure 14: Defining $\oplus$

The following instructions are for p_i.
at time slot 0 **do**:
  Initialize
  Set $\texttt{send} = \texttt{true}$
  $\forall j$, $\texttt{vote}_j = \texttt{true}$, $d_j = 1$

at time slot t **if** $\texttt{send} = \texttt{true}$:
  Collect txns
  If $|\texttt{block-txns}| = B$ **do**: ▷ B as specified in Appendix F
    Set md = metadata for the current block
    Set $\texttt{sndbuf} = \texttt{sndbuf}||\text{md}$ ▷ d_i included in metadata
    Set $\texttt{send} = \texttt{false}$

at every time slot t **do**:
  Let $I = \{j : \texttt{vote}_j = \texttt{true}$ **and** p_i has received a first block b_j of depth d_j from $p_j\}$
  Set $\texttt{sndbuf} = \texttt{sndbuf}|| \oplus_I (\text{VOTE}, j, d_j, H(b_j))$
  $\forall j \in I$ set $\texttt{echo}_j = \texttt{false}$
  $\forall j$ s.t. p_i has received $(\text{VOTE}, j, d_j, H(b'_j))$ from $n - f$ distinct processors (for some fixed b'_j)
  and s.t. p_i has also received b'_j **do**:
    Deliver b'_j
    Set $\texttt{vote}_j = \texttt{true}$, $d_j = d_j + 1$
    If $j = i$, set $\texttt{send} = \texttt{true}$
  Transfer to upload ▷ See Figure 9

Figure 15: CB by all processors

vote sent to n processors. The time between each block delivery is then:

$$\frac{(B + M)n}{S} + \frac{n^2\lambda}{S} + 2\Delta.$$

Since D/n transactions arrive at the client processor of p_i at each timeslot, for latency to be bounded we require:

$$B \geq \frac{D((B + M)n + n^2\lambda + 2\Delta S)}{Sn}.$$

This means:

$$B \geq \frac{M + \lambda n + 2\Delta S/n}{(S/D) - 1}. \tag{15}$$

Since latency will clearly be minimized by minimizing B subject to this constraint, set B equal to the r.h.s. above.

Consider next what happens between time slot 0 and the first timeslot, t_1 say, at which p_i adds metadata to sndbuf. At each timeslot in this interval, less than S/n transactions arrive at the client processor of p_i (since $S > D$), and are immediately added to sndbuf (via the instruction "Collect $txns$"). Each is then added to p_i's upload buffer n times (once for each recipient, via the instruction "Transfer to upload"), with the upload buffer emptied of addressed transactions by the end of the timeslot. It follows that, by the end of time slot t_1, sndbuf does not contain any transactions and that the leader's upload buffer does not contain any addressed transactions.

Now suppose that, at some timeslot t, p_i completes a block, i.e., while its local value send $=$ true, it adds the metadata for some block to its sndbuf. Suppose (inductively) that, by the end of timeslot t, its sndbuf does not contain any transactions and that its upload buffer does not contain any addressed transactions (i.e., all such parcels added to these buffers have been removed). All correct processors will then deliver the next block at time:

$$t' := t + \frac{Mn}{S} + \frac{n^2\lambda}{S} + 2\Delta = t + \frac{nB(S-D)}{DS}.$$

In the interval $(t, t']$, p_i will therefore receive $B(S-D)/S$ many transaction parcels, which will be immediately added to sndbuf at t'. To complete the block, p_i requires a further

$$B - B(S-D)/S = BD/S$$

transaction parcels, which will arrive in time nB/S, which is exactly the time that p_i requires to clear sndbuf of all transactions. It therefore holds that when p_i completes the next block at $t'' := t' + nB/S$, its sndbuf does not contain any transactions and that its upload buffer does not contain any addressed transactions.

To calculate the latency, we have to consider the time from when transactions start building up at t until the next block (which p_i starts sending at t') is delivered by all correct processors. This is:

$$\frac{Bn + 2Mn + 2n^2\lambda}{S} + 4\Delta.$$

As claimed, substituting in the value for B in (15) gives latency:

$$\left(\frac{2Mn + 2\lambda n^2}{S} + 4\Delta\right)\left(1 + \frac{1}{2((S/D)-1)}\right).$$

We may also observe that increasing B increases latency according to the calculations above, so that B as specified by the r.h.s. of (15) minimizes latency.

Appendix G. Reliable Broadcast (single sender)

The protocol is shown in Figure 16.

There is a designated processor p_ℓ. The following instructions are for p_i.
at time slot 0 do:
  Initialize ▷ As specified in Figure 9
  Set $d = 1$, send $=$ true, echo $=$ true, vote $=$ true
    ▷ d is block depth

at time slot t **if** send $=$ true **and** $p_i = p_\ell$:
  Collect txns ▷ As specified in Figure 9
  If |block-txns| $= B$ **do**:
    ▷ B as specified in Section 4
    Set md = metadata for the current block
    Set sndbuf $=$ sndbuf||md
      ▷ d included in metadata
    Set send $=$ false

at every time slot t **do**:
  If echo $=$ true **and** p_i has received a first block b of depth d from p_ℓ **do**:
    Set sndbuf $=$ sndbuf||(ECHO, d, b)
    Set echo $=$ false
  If vote $=$ true **do**:
    If p_i has received (ECHO, d, b') from $n-f$ distinct processors (for some fixed b') **do**:
      Set sndbuf $=$ sndbuf||(VOTE, $d, H(b')$)
        ▷ H a hash function
      Set vote $=$ false
  If vote $=$ true **do**:
    If p_i has received (VOTE, $d, H(b')$) from $f+1$ distinct processors (for some fixed b') **do**:
      Set sndbuf $=$ sndbuf||(VOTE, $d, H(b')$)
      Set vote $=$ false
  If p_i has received (VOTE, $d, H(b')$) from $n-f$ distinct processors (for some fixed b')
  and if p_i has also received b' **do**:
    Deliver b'
    Set send $=$ true, echo $=$ true, vote $=$ true, $d = d+1$
  Transfer to upload ▷ As specified in Figure 9

Figure 16: Reliable Broadcast by a designated sender

Appendix H. Reliable Broadcast (multi-sender)

We assume the reader is familiar with Bracha's broadcast in what follows and analyze only the resulting latency, rather than re-verifying correctness. Every processor receives D/n transaction parcels at its client processor at each time slot and simultaneously reliably broadcasts to all processors. The protocol is shown in Figure 17.

In analysing latency for the protocol of Figure 17, we make the following simplification: if a block b contains B transaction parcels and metadata of size M, then we also suppose that the message (ECHO, d, b) is of size $B + M$. We suppose that *votes*, i.e., messages of the form (VOTE, $d, H(b)$), are of a fixed size λ and analyse latency in

The following instructions are for p_i.
at time slot 0 **do**:
  Initialize
  Set send = true
  $\forall j$, $\texttt{echo}_j$ = true, $\texttt{vote}_j$ = true, $d_j = 1$

at time slot t **if** send = true:
  Collect txns
  If |block-txns| $= B$ **do**: ▷ B as specified in Appendix H
    Set md = metadata for the current block
    Set sndbuf = sndbuf||md ▷ d_i included in metadata
    Set send = false

at every time slot t **do**:
  Let $I = \{j :$ $\texttt{echo}_j$ = true **and** p_i has received a first block b_j of depth d_j from $p_j\}$
  Set sndbuf = sndbuf|| $\oplus_I$ (ECHO, j, d_j, b_j)
  $\forall j \in I$ set $\texttt{echo}_j$ = false
  Let $I = \{j :$ $\texttt{vote}_j$ = true **and** p_i has received (ECHO, j, d_j, b'_j) from $n - f$ distinct processors (for some fixed b'_j)$\}$
  Set sndbuf = sndbuf|| $\oplus_I$ (VOTE, $j, d_j, H(b'_j)$)
  $\forall j \in I$ set $\texttt{vote}_j$ = false
  Let $I = \{j :$ $\texttt{vote}_j$ = true **and** p_i has received (VOTE, j, d_j, b'_j) from $f + 1$ distinct processors (for some fixed b'_j)$\}$
  Set sndbuf = sndbuf|| $\oplus_I$ (VOTE, $j, d_j, H(b'_j)$)
  $\forall j \in I$ set $\texttt{vote}_j$ = false
  $\forall j$ s.t. p_i has received (VOTE, $j, d_j, H(b'_j)$) from $n - f$ distinct processors (for some fixed b'_j)
  and s.t. p_i has also received b'_j **do**:
    Deliver b'_j
    Set $\texttt{echo}_j$ = true, $\texttt{vote}_j$ = true, $d_j = d_j + 1$
    If $j = i$, set send = true
  Transfer to upload

Figure 17: Reliable Broadcast by all processors

the case that all processors are synchronized (i.e., all begin the protocol at the same time) and are correct.

The calculation to establish latency is very similar to that for RB in the single-sender setting. Suppose $S/Dn > 1$. Consider an arbitrary processor p_i and suppose all processors are correct. At each depth, p_i sends a block to all processors. Then all processors echo n blocks (one for each processor) to all processors. Then all processors send votes for each of the n blocks to each of the other processors. The time to deliver p_i's block is therefore at least:

$$\frac{(B+M)n + (B+M)n + n^2\lambda}{S} + 3\Delta.$$

Since D/n transaction parcels arrive at p_i's client processor at each timeslot, for latency to be bounded we require:

$$B \geq \frac{D(B + M + (B+M)n + n\lambda + 3\Delta S/n)}{S}.$$

This means we require B to be greater than or equal to:

$$\frac{D(M + (M+\lambda)n + 3\Delta S/n)}{S - D - Dn}. \tag{16}$$

So, to minimize latency, set B equal to this value.

Consider next what happens between timeslot 0 and the first timeslot, t_1 say, at which any processor p_i adds metadata to sndbuf. At each timeslot in this interval, less than S/n transactions arrive at the client processor of p_i, and are immediately added to sndbuf. Each is then added to p_i's upload buffer n times (once for each recipient), with the upload buffer emptied of addressed transactions by the end of the timeslot. It follows that, by the end of time slot t_1, each processor's sndbuf does not contain any transactions and that each processor's upload buffer does not contain any addressed transactions.

Now suppose that, at some timeslot t, some processor p_i completes a block, i.e., while its local value send = true, it adds the metadata for some block to its sndbuf. Suppose (inductively) that, by the end of timeslot t, its sndbuf does not contain any transactions and that its upload buffer does not contain any addressed transactions (i.e., all such parcels added to these buffers have been removed). All correct processors will then deliver the block at time:

$$\begin{aligned} t' &:= t + \frac{Mn + (B+M)n^2 + n^2\lambda}{S} + 3\Delta \\ &= t + Bn(S - D - Dn)/(DS) + Bn^2/S. \end{aligned}$$

In the interval $(t, t']$, p_i will therefore receive $B(S - D - Dn)/S + BnD/S$ many transaction parcels, which will be immediately added to sndbuf at t'. To complete the block, p_i requires a further

$$B - B(S - D - Dn)/S - BnD/S = BD/S$$

transaction parcels, which will arrive in time nB/S, which is exactly the time that p_i requires to clear sndbuf of all transactions. It therefore holds that when the next layer of blocks is delivered at $t'' := t' + nB/S$, each processor's sndbuf does not contain any transactions and that its upload buffer does not contain any addressed transactions.

To calculate the latency, we have to consider the time from when transactions start building up at t until the next layer of blocks (which processors start sending at t') is delivered. This is:

$$\frac{Bn + 2Bn^2 + 2Mn + 2(M+\lambda)n^2}{S} + 6\Delta.$$

Substituting in the value for B in (16), equating $S - D(n+1)$ with $S - Dn$, and removing lower order terms gives the latency below as claimed:

$$\frac{2(M+\lambda)n^2/S + 6\Delta}{(S/Dn) - 1} + \frac{2(M+\lambda)n^2}{S} + 6\Delta.$$

We may also observe that increasing B increases latency according to the calculations above, so that B as specified in (16) minimizes latency.

Further analysis. Of course, DAG-based protocols generally have metadata which is $\Theta(n)$.

Corollary 1. *Set $M + \lambda = n\lambda'$. If $S/Dn > 1$ and the block size B is set to minimize latency, then, once lower order terms are removed, latency for the protocol of Figure 17 is:*

$$\left(\frac{2\lambda' n^3}{S} + 6\Delta\right)\left(1 + \frac{1}{(S/Dn) - 1}\right).$$

We have already observed in Section 4 that, for RB, there is no factor n difference in the latency bottleneck between the single and multi-sender settings. A second observation is that the first term in parentheses in the statement of Corollary 1 is already *cubic* in n in the case that metadata is $\Theta(n)$, while if metadata in the single-sender case is of constant size then the corresponding term in the single-sender case is $O(n)$. Perhaps to avoid these issues, practical instantiations of DAG-based protocols tend to use a form of CB rather than RB for the purposes of block propagation.

Appendix I.
RB with erasure coding (single-sender)

We next consider an erasure coding version of Reliable Broadcast in the single-sender setting. The protocol is based on that described by Cachin and Tessaro [70], which uses a scheme for erasure coding originally described by Rabin [71]. We assume familiarity with those papers in what follows. At a high level, the protocol uses certain basic functionalities:

- An injective function F, which converts a sequence of parcels m into a tuple $F(m) = (f_1(m), \ldots, f_n(m))$, where each $f_i(m)$ consists of $3|m|/n$ parcels.[16] F satisfies the property that m can be recovered from any $f + 1$ of the values $f_1(m), \ldots, f_n(m)$.
- For each $i \in [1, n]$, a function G_i, which converts a sequence of parcels m into a 'fragment' $(i, f_i(m), x, y)$, where:
  - $f_i(m)$ is the i^{th} component of $F(m)$;
  - x is the Merkle root of the Merkle tree with leaves $f_1(m), \ldots, f_n(m)$ (we also refer to x as the 'Merkle root of m'), and;
  - y is a *fingerprint* of $\Theta(\log(n))$ data parcels, which acts as a witness that $f_i(m)$ is the i^{th} leaf of a Merkle tree with root x.
- A *verification* function V such that $V(i, z, x, y) = \mathsf{true}$ iff y is a witness that x is the Merkle root of a Merkle tree with i^{th} leaf z. For a given x, we let V_x be the set of tuples of the form (i, z, x, y) such that $V(i, z, x, y) = \mathsf{true}$.
- A *reconstruction* function R such that, for any x and any set X of $f + 1$ distinct tuples in V_x, X=$\{(i_1, z_1, x, y_1), \ldots, (i_{f+1}, z_{f+1}, x, y_{f+1})\}$ with $i_j \neq i_{j'}$ for $j \neq j'$, $R(X)$ outputs the unique m such that, for each $j \in [1, f + 1]$, $G_{i_j}(m) = (i_j, z_j, x, y_j)$ (so that $z_j = f_{i_j}(m)$) if there exists such, and outputs false otherwise.

16. As in previous sections, we ignore issues of integer rounding for the sake of simplicity.

The resulting protocol is a modification of the protocol for Reliable Broadcast in Figure 16 and is shown in Figure 20. We note that the instructions must now be altered to incorporate the requirement of erasure coding that the the entire block be known before the leader starts sending any part of the block to others. The protocol uses a function $\oplus^*$, which is defined in Figure 18 below, and also uses the function defined in Figure 19. While the protocol decreases latency compared to that in Figure 16 through the use of erasure coding, there remain various ways in which the protocol is non-optimal. For example, the leader waits until one block is delivered before beginning to send data for the next (i.e., there is no use of 'pipelining'). Another inefficiency is that other processors wait until receiving their full 'fragment' $G_i(b)$ of a block b before beginning to echo that fragment to others. We will address these issues in Appendix K.

Suppose that, for each $i \in [1, n]$, m_i is a sequence of data parcels $m_{i,1}, \ldots, m_{i,x_i}$ (where $m_{i,x}$ is undefined for $x > x_i$). For each i and x, let $m^i_{i,x}$ be the addressed data parcel which is $m_{i,x}$ intended for recipient p_i. Then $\oplus^*_{i\in[1,n]} m_i$ is the sequence of addressed data parcels formed recursively as follows:

Set $x^* = \max\{x_i : \ i \in [1, n]\}$
Set m to be the empty sequence.
For $x = 1$ to x^* do:
  For $i = 1$ to n do:
    If $m_{i,x}$ is defined, set $m := m||m^i_{i,x}$
Output m

Figure 18: Defining $\oplus^*$

$R_{\text{val}}(X, d, x, k)$
  If $X < k$, **output** false
  If it does not hold that every message in X is from a different processor p_j and is of the form $(\text{ECHO}, d, j, z', x, y')$ such that $V(j, z', x, y') = \mathsf{true}$;
    output false
  If it holds for any (equivalently all, assuming hashes are unique) $X' \subseteq X$ of size $f+1$ that $R(X') = \mathsf{false}$;
    output false
  Otherwise output true

Figure 19: The function R_{val}

Since the verification of correctness is essentially the same as in [70], we focus on analyzing latency. To analyze the latency, we make the following simplifications. We suppose that *votes*, i.e. messages of the form (VOTE, d, x), are of a fixed size λ (essentially, the size of a hash), and that the block metadata (perhaps a hash and a signature) is of size $c\lambda$ for some small constant c. If a block b contains B transaction parcels and metadata of size $c\lambda$, then we suppose that a message $(d, G_i(b))$ is of size $3(B + c\lambda)/n + \lambda \log(n)$ and a message $(\text{ECHO}, d, G_i(b))$

There is a designated processor p_ℓ. The following instructions are for p_i.

at time slot 0 **do**:
  Initialize
  Set $d = 1$, send $=$ true, echo $=$ true, vote $=$ true

at time slot t **if** send $=$ true **and** $p_i = p_\ell$:
  Party p_i collects *txns*, which is all data parcels received by the client processor but not yet added to block-txns at any previous time slot.
  block-txns $=$ block-txns $\cup \{txns\}$
  If $|$block-txns$| \geq B$ **do**:
    $\triangleright$ B as specified in Appendix I
    Set md = metadata for the current block
    Set $b = txns||$md
    For each $j \in [1, n]$, set $m_j = (d, j, z, x, y)$, where $(j, z, x, y) = G_j(b)$
    Add $\oplus^*_{j\in[1,n]} m_j$ to upload buffer
    block-txns $= \{\}$
    Set send $=$ false

at every time slot t **do**:
  If echo $=$ true **and** p_i has received a first message (d, i, z, x, y) from p_ℓ such that $V(i, z, x, y) =$ true **do**:
    Set sndbuf $=$ sndbuf$||$(ECHO$, d, i, z, x, y)$
    Set echo $=$ false
  If vote $=$ true **do**:
    If there exists x such that p_i has received a set of messages X with $R_{\text{val}}(X, d, x, n - f) =$ true **do**:
      Set sndbuf $=$ sndbuf$||$(VOTE$, d, x)$
      Set vote $=$ false
  If vote $=$ true **do**:
    If there exists x such that p_i has received (VOTE$, d, x)$ from $f + 1$ distinct processors **do**:
      Set sndbuf $=$ sndbuf$||$(VOTE$, d, x)$
      Set vote $=$ false
  If there exists x such that:
  (i) p_i has received (VOTE$, d, x)$ from $n - f$ distinct processors, and;
  (ii) p_i has received a set of messages X with $R_{\text{val}}(X, d, x, f + 1) =$ true; then **do**:
      Deliver $R(X)$
      Set send $=$ true, echo $=$ true, vote $=$ true, $d = d + 1$
  Transfer to upload

Figure 20: RB using erasure coding with a designated sender

is also of size $3(B + c\lambda)/n + \lambda \log(n)$. We analyse latency in the case that all processors are correct.

Claim 6. *Suppose all processors are correct. If $S > 6D$, then setting blocksize B to minimise latency (and after removing some small terms), the protocol of Figure 20 has latency:*

$$\left(\frac{4n\log(n)\lambda}{S} + 6\Delta\right)\left(1 + \frac{1}{(S/6D) - 1}\right).$$

Proof. Note that, at the first time slot at which the leader completes a block, i.e., transfers metadata to its upload buffer, its upload buffer will be empty prior to this transfer. Set t to be any time slot at which the leader completes a block b and suppose (inductively) that its upload buffer was empty prior to this transfer of data. All correct processes will then send an ECHO message corresponding to b by time $t + 3(B + c\lambda)/S + \lambda n \log(n)/S + \Delta$, by which time the leader's upload buffer will also be empty. All correct processes will then send a VOTE message for b by time:

$$t + 6(B + c\lambda)/S + 2n\log(n)\lambda/S + 2\Delta,$$

by which time the upload buffers of all correct processors will be empty. All correct processes will then deliver the block by time:

$$t' = t + 6(B + c\lambda)/S + n\lambda(2\log(n) + 1)/S + 3\Delta, \quad (17)$$

by which time the upload buffers of all correct processors will be empty. Since D transactions arrive at the client processor of the leader at every time slot, for latency to be bounded, we require:

$$B \geq \frac{D(6(B + c\lambda) + \lambda n(2\log(n) + 1) + 3\Delta S)}{S}.$$

This is equivalent to B being greater than:

$$\frac{D(6c\lambda + \lambda n(2\log(n) + 1) + 3\Delta S}{S - 6D} \quad (18)$$

So, set B equal to the expression in (18) above. For this value of B, the leader will then able to complete the next block b' immediately upon all correct processors delivering b at t'. The latency is then $2(t' - t)$, i.e., $(t' - t)$ to deliver the current block and $(t' - t)$ time to deliver the previous block, during which time transactions for the current block are queued. Approximating $6c+n(2\log(n)+1)$ as $2n\log(n)$ (which will be reasonable so long as n is not small), this gives a latency of

$$\frac{12B + 4n\log(n)\lambda}{S} + 6\Delta.$$

Substituting in the value for B, and again approximating $6c+n(2\log(n)+1)$ as $2n\log(n)$, gives the latency as stated in the claim. Note also that increasing B increases latency in the calculations above, so that our choice of B minimises latency. □

Appendix J.
RB with erasure coding (multi-sender)

We next consider an erasure coding version of Reliable Broadcast in the multi-sender setting. The protocol uses a modified form of R_{val}, shown in Figure 21, and appears in Figure 22.

To analyse the latency, we make similar simplifications to those used in Appendix I. We suppose that *votes*, i.e., messages of the form (VOTE$, j, d, x)$, are of a fixed size λ, and that block metadata is of size M. If a block b contains B

$R^*_{\text{val}}(X, j', d, x, k)$
If $X < k$, **output** false
If it does not hold that every message in X is from a different processor p_j and is of the form $(\text{ECHO}, j', d, j, z', x, y')$ such that $V(j, z', x, y') =$ true;
output false
If it holds for any (equivalently all, assuming hashes are unique) $X' \subseteq X$ of size $f+1$ that $R(X') =$ false;
output false
Otherwise output true

Figure 21: The function R^*_{val}

transaction parcels and metadata of size M, then we suppose that a message $(d, G_j(b))$ is of size $3(B+M)/n+\lambda\log(n)$ and a message $(\text{ECHO}, i, d, G_j(b))$ is also of size $3(B + M)/n + \lambda\log(n)$. We analyse the latency in the case that all processors are synchronized (i.e., all begin the protocol at precisely the same time) and correct.

Claim 7. *Suppose all processors are synchronized and correct. If $S > 3D$, then, setting blocksize B to minimise latency and after removing lower order terms, the protocol of Figure 20 has latency:*

$$\left(\frac{6nM + 2n^2\log(n)\lambda}{S} + 6\Delta\right)\left(1 + \frac{1}{(S/3D) - 1}\right).$$

Proof. Note that, at the first time slot at which any processor completes a block, i.e. transfers metadata to its upload buffer, its upload buffer will be empty prior to this transfer. Now let t be any time slot at which a processor p_i completes a block b and suppose (inductively) that its upload buffer was empty prior to this transfer of data. All correct processes will then send an ECHO message corresponding to b by time $t + 3(B + M)/S + \lambda n\log(n)/S + \Delta$, by which time the leader's upload buffer will also be empty. All correct processes will then send a VOTE message for b by time:

$$t + \tfrac{3(B+M)}{S} + \tfrac{\lambda n\log(n)}{S} + \tfrac{3n(B+M)}{S} + \tfrac{n^2\lambda\log(n)}{S} + 2\Delta,$$

by which time the upload buffers of all correct processors will be empty. All correct processes will then deliver the block by time:

$$t' = t + \tfrac{3(B+M)}{S} + \tfrac{\lambda n\log(n)}{S} + \tfrac{3n(B+M)}{S} + \tfrac{n^2\lambda\log(n)}{S} + \tfrac{n^2\lambda}{S} + 3\Delta,$$

by which time the upload buffers of all correct processors will be empty. Recall that each process receives D/n transactions at its client processor at each time slot. For latency to be bounded we therefore require:

$$B \geq \tfrac{D(3(B+M)+\lambda n\log(n)+3n(B+M)+n^2\lambda\log(n)+n^2\lambda+3\Delta S)}{Sn},$$

which means B must be greater than or equal to:

$$\frac{D(3(n+1)M + n(n+1)(\log(n)+1)\lambda + 3\Delta S)}{Sn - 3(n+1)D}. \quad (19)$$

So, set B equal to the expression in (19) above. Setting B to this value, the leader will be able to complete the next

The following instructions are for p_i.
at time slot 0 **do**:
Initialize
Set send = true
$\forall j$, $\text{echo}_j =$ true, $\text{vote}_j =$ true, $d_j = 1$

at time slot t **if** send = true **do**:
Party p_i collects *txns*, which is all data parcels received by the client processor but not yet added to block-txns at any previous time slot.
block-txns = block-txns $\cup \{txns\}$
If |block-txns| $\geq B$ **do**:
▷ B as specified in Appendix J
Set md = metadata for the current block
Set b = *txns*||md
For each $j \in [1, n]$, set $m_j = (d_i, j, z, x, y)$, where $(j, z, x, y) = G_j(b)$
Add $\oplus^*_{j\in[1,n]} m_j$ to upload buffer
block-txns = $\{\}$
Set send = false

at every time slot t **do**:
Let $I = \{j :$ $\text{echo}_j =$ true **and** p_i has received a first message of the form (d_j, i, z_j, x_j, y_j) (for some z_j, x_j, y_j) from p_j with $V(i, z_j, x_j, y_j) =$ true$\}$
Set sndbuf = sndbuf$|| \oplus_I (\text{ECHO}, j, d_j, i, z_j, x_j, y_j)$
$\forall j \in I$ set $\text{echo}_j =$ false
Let $I = \{j :$ $\text{vote}_j =$ true **and** there exists x_j s.t. p_i has received a set of messages X_j with $R^*_{\text{val}}(X_j, j, d_j, x_j, n - f) =$ true$\}$
Set sndbuf = sndbuf$|| \oplus_I (\text{VOTE}, j, d_j, x_j)$
$\forall j \in I$ set $\text{vote}_j =$ false
Let $I = \{j :$ $\text{vote}_j =$ true **and** there exists x_j s.t. p_i has received $(\text{VOTE}, j, d_j, x_j)$ from $f + 1$ distinct processors$\}$
Set sndbuf = sndbuf$|| \oplus_I (\text{VOTE}, j, d_j, x_j)$
$\forall j \in I$ set $\text{vote}_j =$ false
$\forall j$ s.t. there exists x_j for which p_i has received $(\text{VOTE}, j, d_j, x_j)$ from $n - f$ distinct processors **and** s.t. p_i has also received a set of messages X_j with $R^*_{\text{val}}(X_j, j, d_j, x_j, f + 1) =$ true
Deliver $R(X_j)$
Set $\text{echo}_j =$ true, $\text{vote}_j =$ true, $d_j = d_j + 1$
If $j = i$, set send = true
Transfer to upload

Figure 22: RB by all processors using erasure coding

block b' immediately upon all correct processors delivering b at t'. The latency is then $2(t' - t)$. Approximating $n + 1$ as n and $\log(n) + 1$ as $\log(n)$, this gives a latency of

$$\frac{6(B + M)n + 2n^2\log(n)\lambda}{S} + 6\Delta.$$

Substituting in the value for B, and again approximating $n + 1$ as n and $\log(n) + 1$ as $\log(n)$, gives the latency as

stated in the claim. Note that increasing B beyond the value specified in (19) only increases latency. □

Appendix K.
RB with erasure coding (single-sender with pipelining)

Recall that our motivation for studying the latency of (multi-shot versions of) primitives such as Best-effort Broadcast, Consistent Broadcast, and Reliable Broadcast, is (at least partly) so that we can later extend this analysis to determine latency for state-of-the-art SMR protocols. As preparation for that analysis, we next consider a modification of the protocol in Figure 20 that also uses such optimizations. This modified protocol is designed to be as close as possible to DispersedSimplex, so that our analysis here will later carry over with minimal effort. At a high level, the principal modifications are as follows:

- At the cost of reducing fault-tolerance by one, we remove the requirement that the leader echoes block fragments and sends votes. This means that after sending the fragments for one block b, the leader can immediately begin sending fragments for the next block.
- Rather than having other processors wait until receiving a full fragment before they begin the process of echoing that fragment, we have them pass on those data parcels as they arrive. Since they cannot verify that the fragment is well-formed until the entire message is received, they must send a separate message upon receipt of the full fragment, which indicates that they have received the fragment and it is well-formed.
- This separate message is formed using a threshold signature scheme [72], [73] and is a *share* of a *certificate* for the message (ECHO, d, x), where x is the corresponding Merkle root and d is the block depth. We let m_i denote p_i's share of a certificate for m, and suppose that a certificate for m can be constructed from any set of $n - f$ distinct shares. A certificate for m of the form $m = (\text{ECHO}, d, x)$ is called an echo-certificate for x (of depth d).
- To make the protocol as close as possible to DispersedSimplex, we also use a threshold signature scheme for votes. A certificate for m of the form $m = (\text{VOTE}, d, x)$ is called a vote-certificate for x (of depth d).

Some further details. To implement the above, while ensuring that the protocol remains as similar as possible to DispersedSimplex, we suppose that each processor maintains a *certificate pool* C. Whenever a processor receives at least $n - f$ shares of some echo/vote-certificate (and if C does not already contain the corresponding certificate), it generates a certificate, adds it to C, and sends the certificate to all processors. Similarly, whenever a processor receives a certificate that does not already belong to C, it enumerates it into C and sends that certificate to all processors.

The protocol, which uses the procedures defined in Figure 23, is shown in Figure 24.

Latency. To analyse the latency, we make similar simplifications to those used previous sections. We suppose that *votes*, i.e., messages of the form $(\text{VOTE}, d, x)_i$, are of a fixed size λ. Similarly, echo-messages, i.e., messages of the form $(\text{ECHO}, d, x)_i$, are of a fixed size λ, and vote/echo-certificates are of size λ.[17] If a block b contains B transaction parcels and metadata of size M, then we suppose that a message $(d, G_j(b))$ is of size $3(B+M)/n+\lambda \log(n)$. We analyse latency in the case that all processors are correct.

$R^{\dagger}_{\text{val}}(X, d, x)$
If $X \neq f + 1$, **output** false
If it does not hold that every message in X is from a different processor p_j and is of the form (d, j, z', x, y') such that $V(j, z', x, y') = \text{true}$;
output false
If it holds that $R(X) = \text{false}$;
output false
Otherwise output true

Update certificates
For any x and d' such that p_i has received at least $n-f$ shares of an echo/vote-certificate for x of depth d', construct the corresponding certificate and enumerate it into C (if not already in C).
For any x and d' such that such that p_i has received an echo/vote-certificate for x of depth d', enumerate that certificate into C (if not already in C).

Send certificates
For each certificate $c \in$ C that has not previously been added to sendbuffer:
Set sndbuf := sndbuf$||c$

Gossip from leader
For each data parcel m received from p_ℓ and not yet added to sendbuffer:
sndbuf := sndbuf$||m$

Figure 23: $R^{\dagger}_{\text{val}}$, 'Update certificates', 'Send certificates' and 'Gossip from leader'

To calculate latency when B is well chosen, we first consider how the protocol is intended to function. For some time interval L, and for $t_i := iL$, the idea is that the leader will disseminate fragments for some first block b_1 from t_1 until t_2, and will disseminate fragments for a second block b_2 from t_2 until t_3, and so on. Other processors will begin

17. We assume that threshold signatures are used to create these certificates. In practice, there is a trade-off between the amount of computation and communication. Threshold signatures such as BLS will incur high computation per signature verification but requires only one signature verification for an n-sized certificate. The signature size is smaller (λ-sized) as well. On the other hand, Ed25519 will incur lower computation per signature verification but an n-sized certificate consists of n signatures, each of which is λ-sized.

There is a designated processor p_ℓ. The following instructions are for p_i.

at time slot 0 **do**:
  Initialize, set $d = 1$, $\mathtt{C} = \{\}$, for all d' set
  $\mathtt{echo}(d') = \mathsf{true}$, $\mathtt{vote}(d') = \mathsf{true}$

at time slot t **if** $p_i = p_\ell$:
  Party p_i collects *txns*, which is all data parcels received by the client processor but not yet added to block-txns at any previous time slot.
  block-txns = block-txns $\cup \{txns\}$
  If $|$block-txns$| \geq B$ **do**:
    ▷ B as specified in Appendix K
    Set md = metadata for the current block
    Set $b = txns||$md
    For each $j \in [1, n]$, set $m_j = (d, j, z, x, y)$, where $(j, z, x, y) = G_j(b)$
    Add $\oplus^*_{j\in[1,n]} m_j$ to upload buffer
    block-txns $= \{\}$, $d := d + 1$

at every time slot t **if** $p_i \neq p_\ell$ **do**:
  Update certificates ▷ As specified in Figure 23
  If $\mathtt{echo}(d) = \mathsf{true}$ **and** p_i has received a first (full) message (d, i, z, x, y) from p_ℓ such that $V(i, z, x, y) = \mathsf{true}$ **do**:
    Set sndbuf $=$ sndbuf$||$(ECHO$, d, x)_i$
    ▷ Send share of echo-certificate
    Set $\mathtt{echo}(d) = \mathsf{false}$, $d := d + 1$
  For all $d' \leq d$, **if** $\mathtt{vote}(d') = \mathsf{true}$ **do**:
    If $\exists x$ s.t. C contains an echo certificate for x of depth d' **and** p_i has received a set of messages X with $R^\dagger_{\text{val}}(X, d', x) = \mathsf{true}$ **do**:
    ▷ $R^\dagger_{\text{val}}$ as specified in Figure 23
      Set sndbuf $=$ sndbuf$||$(VOTE$, d', x)_i$
      Set $\mathtt{vote}(d') = \mathsf{false}$
  For all $d' \leq d$, **if** $\mathtt{vote}(d') = \mathsf{true}$ **do**:
    If $\exists x$ s.t. p_i has received $f + 1$ signature shares of a vote-certificate for x of depth d' **do**:
      Set sndbuf $=$ sndbuf$||$(VOTE$, d', x)_i$
      Set $\mathtt{vote}(d') = \mathsf{false}$
  For all x and $d' \leq d$ such that:
  (i) C contains a vote-certificate for x of depth d', and;
  (ii) p_i has received a set of messages X with $R^\dagger_{\text{val}}(X, d', x) = \mathsf{true}$, **do**:
    Deliver $R(X)$ (if not already delivered)
  Send certificates ▷ As specified in Figure 23
  Gossip from leader ▷ As specified in Figure 23
  Transfer to upload ▷ As specified in Figure 9

Figure 24: RB using erasure coding and pipelining with a designated sender

to receive fragments of b_i at $t_i + \Delta$, will immediately start gossiping those fragments on to other processors, and will receive the full fragments by $t_{i+1} + \Delta$. The hope is that, at this time, they will be ready to start gossiping fragments for b_{i+1}, i.e. doing so will not lead to an increasing backlog at their sendbuffer. However, we must take into account that, during the interval between $t_i + \Delta$ and $t_{i+1} + \Delta$ (and in the steady state), processors other than p_ℓ will also have to send to all processors: (i) an echo-message for some block, (ii) an echo-certificate for some block, (iii) a vote for some block, and (iv) a vote-certificate for some block. If B transactions are included in each block, then clearing these messages from sndbuf, together with all the fragments of b_i that they must gossip, takes time at least:

$$L := \frac{3(B + M) + n\lambda \log(n) + 4n\lambda}{S}. \tag{20}$$

This means we are sending data at a rate of at most:

$$\frac{BS}{3(B + M) + n\lambda \log(n) + 4n\lambda}.$$

For latency to be bounded, we require this rate to be at least D, meaning B is at least:

$$\frac{D(3M + n\lambda(\log(n) + 4))}{S - 3D}. \tag{21}$$

Set B equal to this value. In this case, it takes precisely L time slots for the transactions in each block to arrive at the client processor of p_ℓ, so that p_ℓ can indeed start sending b_i at t_i (for $i \geq 1$). It takes time $(3(B + M) + n\lambda \log(n))/S$ for the leader to clear the fragments for each block from its upload buffer, meaning that each of the other processors is able to clear from their upload buffers the corresponding fragment of each block, as well as the required messages of types (i)-(iv) above, in time at most $(3(B + M) + n\lambda \log(n))/S + 4n\lambda/S = L$.

Finally, to calculate latency, note that the first transactions in b_i to arrive at the client processor of p_ℓ do so at t_{i-1}. Correct processors add echo-messages for b_i to their sendbuffer at $t_{i+1} + \Delta$, which are cleared from their upload buffer[18] by $t_{i+1} + \Delta + n\lambda/S$ and are received by all other processors by $t_{i+1} + n\lambda/S + 2\Delta$. Correct processors add votes for b_i to their sendbuffer at this time, which are cleared from their sendbuffer and received by all correct processors by time $t_{i+1} + 2n\lambda/S + 3\Delta$, whereupon they deliver the block. This gives a total latency of $2L + 2n\lambda/S + 3\Delta$, which is:

$$\left(\tfrac{6M+2n\lambda(\log(n)+4)}{S}\right) \cdot \left(1 + \tfrac{1}{(S/3D)-1}\right) + 2n\lambda/S + 3\Delta.$$

This justifies the following claim:

Claim 8. *Suppose all processors are correct. If $S > 3D$, then setting blocksize B to minimise latency, the protocol of Figure 24 has latency:*

$$\left(\tfrac{6M+2n\lambda(\log(n)+4)}{S}\right) \cdot \left(1 + \tfrac{1}{(S/3D)-1}\right) + 2n\lambda/S + 3\Delta.$$

18. Here we make the mild assumption that the timeslots at which processors clear echo-messages, votes and certificates from their sendbuffer are disjoint. For some input parameters this may not be true, giving an increase in latency bounded by $6n\lambda/S$ (and where this bound could be reduced to $n\lambda/S$ by prioritising the sending of echo-messages first, then votes, then certificates).

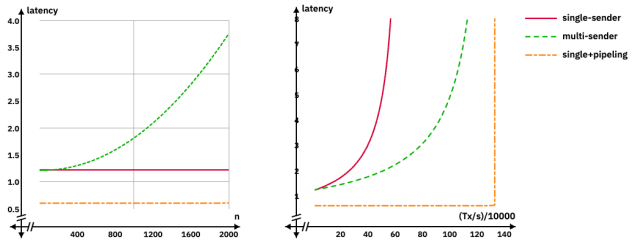

Figure 25: Latency for RB with erasure coding: parameters are explained in Section K.1

K.1. Comparing latency for Reliable Broadcast protocols with erasure coding

To illustrate the trade-offs between the different settings, we suppose again that processors can upload/download at a rate of 10 Gbps. We suppose transactions are 2500 bits (about 300 bytes, similar to typical Bitcoin transactions). In the single-sender setting, we suppose M is 1000 bits, while in the multi-sender setting, we suppose $M = 500 + 500n$ bits. We set $\lambda = 500$ bits and $\Delta = 0.2$ seconds. The first graph in Figure 25 supposes the incoming transaction rate D is 10000 transactions per second and shows the resulting latency (in seconds) as a function of n. The second graph (on the right) in Figure 25 fixes $n = 200$ and shows the resulting latency as a function of the number of incoming transactions per second divided by 10000.

Appendix L. Latency Analysis for Tendermint

In this section, we analyze latency for Tendermint. Since there are multiple versions of the protocol, and for concreteness, we consider the version of Tendermint described in Figure 26. It is not our aim here to *explain* the Tendermint protocol, and so it would be a distraction to make all of the instructions entirely explicit (e.g., how a processor determines whether a block proposal received from the leader is valid, or what it means for a processor to 'set their lock'): the figure is intended only to specify when messages are sent, and our analysis below will also specify message sizes. The version of the protocol described in Figure 26 does not make use of any optimizations such as pipelining.

To calculate latency in the single-sender setting, we consider the case that there is a single processor p_ℓ which is the leader of every view, i.e., $p_\ell = \texttt{lead}(v)$ for all v. Since the protocol does not make any optimizations that take advantage of a stable leader this calculation will also reflect latency in the case of rotating leaders. We suppose 'votes' (stage 1 or 2) are of size λ. For the sake of simplicity, we also suppose that 'locks' are of size λ (the use of a threshold signature scheme allowing for locks of approximately the size of a signature).

Let C be the clearing time, as defined in Appendix B. If p_ℓ enters view v at t, then it will complete a block for view v

The following instructions are for p_i.
at time slot 0 do:
  Initialize
  Set $v = 1$, $\texttt{starttime}(1) = 0$, $\texttt{send} = \texttt{true}$,
  $\texttt{vote1} = \texttt{true}$, $\texttt{vote2} = \texttt{true}$

at time slot t **if** $\texttt{send} = \texttt{true}$ **and** $p_i = \texttt{lead}(v)$ **and** $t - \texttt{starttime}(v) = C(\lambda) + 2\Delta$ **do**:
  ▷ Leader waits for lock info; λ as specified in Section L
  Collect txns ▷ As specified in Figure 9
  If |block-txns| $\geq B$ **do**:
    ▷ B as specified in Section L
    Set md = metadata for the current block
    Set sndbuf = sndbuf||md
    Set send = false

at every time slot t **do**:
  If $\texttt{vote1} = \texttt{true}$ **and** p_i has received a valid block proposal for view v from $\texttt{lead}(v)$ **do**:
    Set sndbuf = sndbuf||(1-VOTE, $H(b)$)$_i$
    ▷ Stage 1 vote on hash of b signed by p_i
    Set $\texttt{vote1} = \texttt{false}$
  If $\texttt{vote2} = \texttt{true}$ **and** p_i has received a 1-QC for a view v block b **do**:
    Set sndbuf = sndbuf||(2-VOTE, $H(b)$)$_i$
    ▷ Stage 2 vote on hash of b signed by p_i
    Set $\texttt{vote2} = \texttt{false}$, Set lock
    Set sndbuf = sndbuf||lock
  If p_i has received a 2-QC for a view v block b **do**:
    Deliver b and all ancestor blocks
    Form threshold certificate c for view $v + 1$, set sndbuf = sndbuf||c
    Set $\texttt{send} = \texttt{true}$, $\texttt{vote1} = \texttt{true}$, $\texttt{vote2} = \texttt{true}$, $\texttt{starttime}(v + 1) = t$, $v = v + 1$
  If $t - \texttt{starttime}(v) \geq$ TIMEOUT **do**:
    Set sndbuf = sndbuf||(COMPLAIN, v)$_i$
    ▷ Signed complaint for view v
  If p_i has received a complaint-QC for view v **do**:
    Form threshold certificate c for view $v + 1$, set sndbuf = sndbuf||c
    Set $\texttt{send} = \texttt{true}$, $\texttt{vote1} = \texttt{true}$, $\texttt{vote2} = \texttt{true}$, $\texttt{starttime}(v + 1) = t$, $v = v + 1$
  Transfer to upload ▷ As specified in Figure 9

Figure 26: Tendermint

(i.e., place the corresponding metadata on sendbuffer) by $t + C(\lambda) + 2\Delta$ at the earliest. If the block includes B transaction parcels and metadata of M parcels, all processors will then send stage 1 votes by $t + C(\lambda) + C(B + M) + 3\Delta$ and stage 2 votes by $t + 2C(\lambda) + C(B + M) + 4\Delta$ at the earliest. Processors therefore receive a 2-QC for the block and enter view $v + 1$ by $t + 3C(\lambda) + C(B + M) + 5\Delta$ at the earliest. This means data is sent at a rate of at most:

$$\frac{B}{3C(\lambda) + C(B + M) + 5\Delta}.$$

For latency to be bounded, this must be greater than or equal to D, so:

$$B \geq D\left(\frac{(3\lambda + B + M)n}{S} + 5\Delta\right).$$

This means B is at least:

$$\frac{D((3\lambda + M)n + 5\Delta S)}{S - Dn}. \tag{22}$$

So, set B equal to this value: as in previous sections, one can then easily verify that the bound $3C(\lambda) + C(B + M) + 5\Delta$ on the length of a view calculated above is tight (we suppose TIMEOUT is set to some greater value), except for the first view. Latency must be measured from the time at which a transaction first arrives at the client processor of p_ℓ. For a transaction included in the block for view v, this could be any time after the leader 'completes' the block for view $v - 1$ by adding the corresponding metadata to sendbuffer (the first view is a special case, but accords the bounds below). Latency is therefore:

$$\frac{(5\lambda + 2B + 2M)n}{S} + 8\Delta.$$

Subtituting in the value for B in (22), we concluded that latency is:

$$\frac{(6\lambda n + 2Mn)/S + 10\Delta}{(S/Dn) - 1} + \frac{(5\lambda + 2M)n}{S} + 8\Delta. \tag{23}$$

Appendix M.
Latency Analysis for Hotstuff

The principal aim of HotStuff [7], [8] is to obtain a protocol with linear communication complexity within each view. While HotStuff incurs low communication complexity, all the messages are relayed through the leader and hence it becomes the bottleneck. This has the possibility of impacting the 'real-world' latency. In this section, we use our model to analyze latency for HotStuff.

For concreteness, we consider the version of HotStuff described in Figure 27. As for Tendermint, it is not our aim here to explain the protocol: the figure is intended only to specify when messages are sent, and our analysis below will also specify message sizes. The version of the protocol described in Figure 27 is intended to reflect that in the original paper [7], and does not make use of any optimizations such as pipelining.

As for Tendermint, to calculate latency in the single-sender setting, we consider the case that there is a single processor p_ℓ which is the leader of every view, i.e., $p_\ell = \texttt{lead}(v)$ for all v. Since the protocol does not make any optimizations that take advantage of a stable leader, this calculation will also reflect latency in the case of rotating leaders. We suppose 'NEWVIEW' messages, 'votes' and QCs (stage 1, 2, or 3) are of size λ. We analyze latency in the case that all processors are correct and are synchronized.

The following instructions are for p_i.
at time slot 0 do:
  Initialize
  Set $v = 1$, $\texttt{starttime}(1) = 0$, $\texttt{send} = \texttt{true}$, $\texttt{vote1} = \texttt{true}$, $\texttt{vote2} = \texttt{true}$, $\texttt{vote3} = \texttt{true}$
  Set $\texttt{send1QC} = \texttt{true}$, $\texttt{send2QC} = \texttt{true}$, $\texttt{send3QC} = \texttt{true}$

at every time slot t **do**:

  Add a NEWVIEW message for view v to upload buffer, with all parcels addressed to $\texttt{lead}(v)$, if not already done

  If $p_i = \texttt{lead}(v)$ **and** $\texttt{send} = \texttt{true}$ **and** p_i has received $n - f$ NEWVIEW messages for view v **do**:
    Collect txns ▷ As specified in Figure 9
    If $|\texttt{block-txns}| \geq B$ **do**: ▷ B as specified in Appendix M
      Set md = metadata for the current block
      Set $\texttt{sndbuf} = \texttt{sndbuf}||\text{md}$
      Set $\texttt{send} = \texttt{false}$

  If $\texttt{vote1} = \texttt{true}$ **and** p_i has received a valid block proposal b for view v from $\texttt{lead}(v)$ **do**:
    Set $\texttt{vote1} = \texttt{false}$
    Add $(\text{1-VOTE}, H(b))_i$ to upload buffer, with all parcels addressed to $\texttt{lead}(v)$ ▷ Stage 1 vote on hash of b signed by p_i

  If $p_i = \texttt{lead}(v)$ **and** $\texttt{send1QC} = \texttt{true}$ **and** p_i has received $n - f$ stage 1 votes for b corresponding to view v **do**:
    Set $\texttt{send1QC} = \texttt{false}$
    Form Q_1, a threshold 1-QC for b
    Set $\texttt{sndbuf} = \texttt{sndbuf}||Q_1$

  If $\texttt{vote2} = \texttt{true}$ **and** p_i has received a 1-QC for a view v block b from $\texttt{lead}(v)$ **do**:
    Set $\texttt{vote2} = \texttt{false}$
    Add $(\text{2-VOTE}, H(b))_i$ to upload buffer, with all parcels addressed to $\texttt{lead}(v)$ ▷ Stage 2 vote on hash of b signed by p_i

  If $p_i = \texttt{lead}(v)$ **and** $\texttt{send2QC} = \texttt{true}$ **and** p_i has received $n - f$ stage 2 votes for b corresponding to view v **do**:
    Set $\texttt{send2QC} = \texttt{false}$
    Form Q_2, a threshold 2-QC for b
    Set $\texttt{sndbuf} = \texttt{sndbuf}||Q_2$

  If $\texttt{vote3} = \texttt{true}$ **and** p_i has received Q_2, a 2-QC for a view v block b, from $\texttt{lead}(v)$ **do**:
    Set $\texttt{vote3} = \texttt{false}$
    Set $\texttt{lock} = Q_2$
    Add $(\text{3-VOTE}, H(b))_i$ to upload buffer, with all parcels addressed to $\texttt{lead}(v)$ ▷ Stage 3 vote on hash of b signed by p_i

  If $p_i = \texttt{lead}(v)$ **and** $\texttt{send3QC} = \texttt{true}$ **and** p_i has received $n - f$ stage 3 votes for b corresponding to view v **do**:
    Set $\texttt{send3QC} = \texttt{false}$
    Form Q_3, a threshold 3-QC for b
    Set $\texttt{sndbuf} = \texttt{sndbuf}||Q_3$

  If p_i has received a 3-QC for a view v block b **do**:
    Deliver b and all ancestor blocks
    Set $\texttt{send} = \texttt{true}$, $\texttt{vote1} = \texttt{true}$, $\texttt{vote2} = \texttt{true}$, $\texttt{starttime}(v + 1) = t$, $v = v + 1$
    Set $\texttt{send1QC} = \texttt{true}$, $\texttt{send2QC} = \texttt{true}$, $\texttt{send3QC} = \texttt{true}$

  If $t - \texttt{starttime}(v) \geq$ TIMEOUT **do**:
    Set $\texttt{sndbuf} = \texttt{sndbuf}||(\text{COMPLAIN}, v)_i$
      ▷ Signed complaint for view v
  If p_i has received a complaint-QC for view v **do**:
    Form threshold certificate c for view $v + 1$, set $\texttt{sndbuf} = \texttt{sndbuf}||c$
    Set $\texttt{send} = \texttt{true}$, $\texttt{vote1} = \texttt{true}$, $\texttt{vote2} = \texttt{true}$, $\texttt{starttime}(v + 1) = t$, $v = v + 1$
    Set $\texttt{send1QC} = \texttt{true}$, $\texttt{send2QC} = \texttt{true}$, $\texttt{send3QC} = \texttt{true}$

  Transfer to upload ▷ As specified in Figure 9

Figure 27: HotStuff

Let C be the clearing time, as defined in Appendix B. If

p_ℓ enters view v at t, then it will complete a block for view v (i.e., place the corresponding metadata on sendbuffer) by $t + C(\lambda) + \Delta$ at the earliest. If the block includes B transaction parcels and metadata of M parcels, all processors will then send stage 1 votes by $t + C(\lambda) + C(B + M) + 2\Delta$, stage 2 votes by $t + 3C(\lambda) + C(B + M) + 4\Delta$, and stage 3 votes by $t + 5C(\lambda) + C(B + M) + 6\Delta$ at the earliest. All processors then receive a stage 3 QC (and enter view $v + 1$) by $t + 7C(\lambda) + C(B + M) + 8\Delta$ at the earliest. This means data is sent at a rate of at most:

$$\frac{B}{7C(\lambda) + C(B + M) + 8\Delta}.$$

For latency to be bounded, this must be greater than or equal to D, so:

$$B \geq D\left(\frac{(7\lambda + B + M)n}{S} + 8\Delta\right).$$

This means B is at least:

$$\frac{D((7\lambda + M)n + 8\Delta S)}{S - Dn}. \tag{24}$$

So, set B equal to this value: as in previous sections, one can then easily verify that the bound $7C(\lambda) + C(B + M) + 8\Delta$ on the length of a view calculated above is tight (we suppose TIMEOUT is set to some greater value), except for the first view. Latency must be measured from the time at which a transaction first arrives at the client processor of p_ℓ. For a transaction included in the block for view v, this could be any time after the leader 'completes' the block for view $v - 1$ by adding the corresponding metadata to sendbuffer (the first view is a special case, but accords the bounds below). Latency is therefore:

$$\frac{(13\lambda + 2B + 2M)n}{S} + 15\Delta.$$

Subtituting in the value for B in (24), we concluded that latency is:

$$\frac{(14\lambda n + 2Mn)/S + 16\Delta}{(S/Dn) - 1} + \frac{(13\lambda + 2M)n}{S} + 15\Delta. \tag{25}$$

Comparing with equation (23), we see that latency for HotStuff is strictly greater than latency for Tendermint, for all parameter values.