Optimal Aggregation via Overlay Trees: Delay-MSE Tradeoffs under Failures

PARIKSHIT HEGDE and GUSTAVO DE VECIANA, The University of Texas at Austin, USA

Many applications, e.g., federated learning, require the aggregation of information across a large number of distributed nodes. In this paper, we explore efficient schemes to do this at scale leveraging aggregation at intermediate nodes across overlay trees. Files/updates are split into chunks which are in turn simultaneously aggregated over different trees. For a synchronous setting with homogeneous communications capabilities and deterministic link delays, we develop a delay optimal aggregation schedule. In the asynchronous setting, where delays are stochastic but i.i.d., across links, we show that for an asynchronous implementation of the above schedule, the expected aggregation delay is near-optimal. We then consider the impact that failures in the network have on the resulting Mean Square Error (MSE) for the estimated aggregates and how it can be controlled through the addition of redundancy, reattempts, and optimal failure-aware estimates for the desired aggregate. Based on the analysis of a natural model of failures, we show how to choose parameters to optimize the trade-off between aggregation delay and MSE. We present simulation results exhibiting the above mentioned tradeoffs. We also consider a more general class of correlated failures and demonstrate via simulation the applicability of our techniques in those settings as well.

CCS Concepts: • **Networks** → **Application layer protocols**; **Network performance analysis**; • **Mathematics of computing** → Stochastic processes.

Additional Key Words and Phrases: Overlay Aggregation, Redundancy, Optimal Estimation

ACM Reference Format:
Parikshit Hegde and Gustavo de Veciana. 2024. Optimal Aggregation via Overlay Trees: Delay-MSE Tradeoffs under Failures . *Proc. ACM Meas. Anal. Comput. Syst.* 8, 3, Article 41 (December 2024), 37 pages. https://doi.org/10.1145/3700423

1 INTRODUCTION

Information aggregation over networks is a fundamental problem in distributed systems [4, 16, 18, 22, 36]. For example, in the context of Federated Learning, a server needs to obtain the average of the local gradients of a large number of distributed clients in each round of training [29]. It is well known that minimizing the time required to aggregate local gradients over the network is critical to speeding up the training time in such systems [20, 24, 26, 29]. Given the size of clients' local gradients may be upwards of 100s of MBs, optimized structured approaches to aggregation are of great interest.

In this paper, we consider the problem of aggregating *updates* present at a set of clients to a single location which we refer to as the *server*, using a complete overlay network so as to reduce the aggregation delay. Fig. 1 presents a simple illustration of the advantage of overlay aggregation.

This material is based upon work of Hegde and de Veciana supported by the National Science Foundation (NSF) under grant No. 2148224 and is supported in part by funds from OUSD R&E, NIST, and industry partners as specified in the Resilient & Intelligent NextG Systems (RINGS) program and the WNCG/6G@UT industrial affiliates.
Authors' address: Parikshit Hegde, hegde@utexas.edu; Gustavo de Veciana, deveciana@utexas.edu, Chandra Department of Electrical and Computer Engineering, The University of Texas at Austin, Austin, Texas, USA .

ACM 2476-1249/2024/12-ART41
https://doi.org/10.1145/3700423

In the example on the left, all the clients directly send their updates to the server, and the server aggregates, e.g, averages, these updates. In the example on the right, an overlay *aggregation tree* is used where intermediate nodes aggregate the information they receive along with their own updates and forward only the aggregate. Assuming that only one message can be transmitted or received at a time by any node and that each transmission takes 1 unit of time, the direct aggregation approach takes 4 units of time to complete, whereas the overlay aggregation approach takes only 3 units of time. As we will see in the sequel, these benefits grow for larger networks.

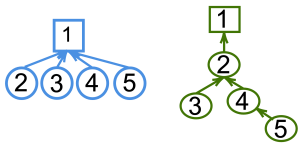


Fig. 1. The example on the left is an example of direct aggregation, and the example on the right is an example of overlay aggregation.

Observe that the example in Fig. 1 does not use all the links available in the complete overlay network. Dividing the updates into *chunks* and aggregating them concurrently using an *aggregation schedule* allows one to make better use of all the links and provides a further speedup leveraging the power of parallelization.

Moreover, since such networks may be operating in unreliable and/or congested conditions, the aggregation of updates may be subject to failure. This means that some of the transmissions scheduled in the network may fail. In this case, the server may fail to obtain the true aggregate of all the clients' updates. In fact an approach to aggregation based on an overlay network may be more drastically affected by failures as compared to direct aggregation. In the examples in Fig. 1, suppose that the transmission from Client 4 fails. In the direct aggregation example, the server still receives the aggregate of Clients 2, 3, and 5. Whereas, in the overlay aggregation example, the server only receives the aggregate of Clients 2 and 3. To address this problem, in this paper, we study structured overlay aggregation where the aim is to average the client's updates and use *redundancy* and a *failure-aware estimation* approach at the server to control tradeoffs amongst aggregation delay and the Mean-Squared-Error (MSE) of the estimated aggregate at the server.

1.1 Contributions

We begin by considering an idealized synchronous communication setting where all link delays are fixed and equal, and design an overlay aggregation schedule, OptSched, that is optimal in terms of minimizing aggregation delay. This in turn permits us to characterize the optimal chunk size as a function of the link delays in the network. A solution to the problem of information dissemination where a server wants to disseminate a file to a set of clients may be viewed as a time-reversal of a solution to the problem of information aggregation that we consider. In this context, Farley [9] proposed a solution to information dissemination that is similar to the time reversal of OptSched under a synchronous setting.

The novel contributions of our paper are as follows.

(1) **Sensitivity of aggregation to stragglers:** We consider a network setting where link delays are random but i.i.d., and show in the case where they are light-tailed that an asynchronous implementation of OptSched is near-optimal in terms of minimizing mean aggregation delay. This is perhaps surprising as one would expect that in a large system, aggregation would face a commensurately large number of straggler links, and hence the performance of an

asynchronous version of OptSched would suffer. Stragglers however will hurt aggregation performance in a system with heavy-tailed link delays. For such systems we show via simulation that terminating scheduled aggregation of updates that are taking too long is effective. Here, the technical novelty of our work lies in developing a bounding process for the aggregation delay of the asynchronous version of OptSched which could be a useful tool in analyzing the performance of asynchronous relaxations of other synchronous scheduling systems.

(2) **Using redundant aggregation to achieve robustness to link failures:** As mentioned above, tree-based aggregation might be viewed as being highly susceptible to link failures e.g., at the top of the tree. One way to address this is to use redundant and diverse aggregation trees. This is not unlike repetition coding and leads to a new class of problems associated with evaluating tradeoffs in the amount of redundancy (and thus aggregation delay) vs reduced MSE in the eventually computed aggregate. We propose a novel joint aggregation and estimation protocol, OptAgg, that extracts an estimate for the aggregate given partial information received across redundant aggregation trees. In a setting where failures are i.i.d., across links, we perform an upper bound analysis based on which we propose design principles to realize desired delay-MSE tradeoffs for OptAgg.

(3) **Performance simulations and comparisons:** Finally, we present a broad set of simulation results assessing the impact of link delay distributions in an i.id., setting as well as in correlated and heterogeneous settings. Our simulations show that our theoretical results still capture the delay-MSE tradeoffs even if the underlying assumptions are weakened. Furthermore, simulations comparing our structured approach to a state-of-the-art gossip algorithm for aggregation show that OptAgg can provide significant aggregation delay improvement for a large range of desired MSE of the aggregate estimate.

1.2 Related Work

Information aggregation is an important problem in systems such as sensor networks where environmental data such as temperature may need to be aggregated from a large set of sensors either at a single point or at a subset of the sensors. These networks may need to operate in highly unreliable conditions and the message sizes are usually small (in Kbs). Therefore, generally, such systems use gossip protocols for aggregation without chunking [6, 8, 12, 16, 17, 22, 23, 28, 34]. However, since we consider the aggregation of larger messages (>100s of Mbs), we propose a structured approach to aggregation instead of a gossip-based approach. Since, in a gossip-based aggregation protocol, a node may receive the update of another node multiple times, Kempe et al.[22] propose a "push-sum" protocol, and Considine et al.[8] use sketching to compute an *unbiased* estimate for the aggregate. In our aggregation with estimation protocol OptAgg we compute an unbiased estimate from the partial information received from redundant copies under failure that also minimizes a bound on the MSE.

Unlike gossip-based algorithms, structured approaches to aggregation have also been studied [5, 27, 30, 36]. Tree-based approaches [27, 36] lead to small aggregation delay as explained in the simple example in Fig. 1, but are susceptible to information loss due to there being a single path from each client to the server. To protect against information loss due to failures, Motegi et al. [30] construct a directed-acyclic graph (DAG) such that there are multiple paths from a client to the server. In our work, we aggregate redundant replicas on different trees and estimate using partial information received from the different replicas to protect against failures. Further, chunking updates and aggregating the chunks simultaneously provides further speedup over using a single tree or DAG without chunking.

Information dissemination is another fundamental problem in distributed systems where information present at one node needs to be disseminated to a set of nodes. Dissemination may be viewed as a complementary problem to aggregation, meaning that to disseminate information one runs an aggregation schedule in reverse time. Farley[9] proposed a similar schedule to the one constructed by our protocol OptSched and proved its optimality for the dissemination problem in the synchronous setting. However, they did not consider an asynchronous implementation method for the schedule nor its analysis. Farley et al.[10] considered the problem of what is the minimum number of edges required in an overlay network to achieve dissemination within a certain time constraint. However, they do not consider the chunking of information. Bar-Noy and Kipnis[2] consider the dissemination problem in a synchronous setting where nodes may send and receive simultaneously. In our work, we only consider the setting where nodes may either send or receive at a time and leave it to future work to adapt our techniques to the other setting. Karp et al.[21] consider dissemination and aggregation problems in the *logP* model where nodes may transmit and receive simultaneously, but they may only initiate communications after certain intervals of initiating or completing a previous communication. Sanghavi et al.[32] consider the problem of information dissemination with chunking in a synchronous setting using a novel gossip algorithm. However, it is not straightforward to perform aggregation using their approach.

In-Network Aggregation has been used in Distributed Machine Learning and Federated Learning where network hardware such as network switches are used to aggregate information en route to the server[11, 25]. This is an alternate approach where the physical layer of the network is used to speed up aggregation. In our work, we focus on an overlay setting where client nodes that themselves generate information are used to speed up aggregation. In the sequel, in-network and overlay aggregation ideas may be combined.

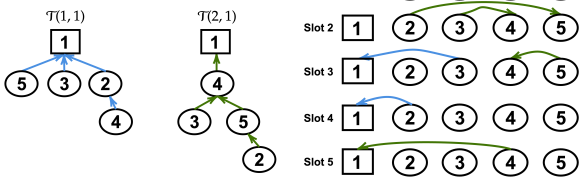
Notation. A set of k elements shall be denoted as $\{x_1, \ldots, x_k\}$, while an ordered set of elements shall be denoted as $(x_1, \ldots, x_k)$. For positive integers $N_1 < N_2$, $[N_1, N_2]$ denotes the set of integers $\{N_1, \ldots, N_2\}$, and $[N_1]$ denotes the set $[1, N_1]$. $\log x$ will denote the base 2 logarithm of x, and $\ln x$ will denote the base e logarithm of x.

2 MODEL SETUP

N nodes are connected over a complete overlay network. $N-1$ of these nodes, labeled as Nodes 2 to N, are called *clients*, and Node 1 is called the *server*. Each Client n has an update $\boldsymbol{u}^n$, which is a vector of dimension dim, and the goal is to compute the *aggregate*, $\boldsymbol{u} \triangleq \frac{1}{N-1}\sum_{n=2}^{N} \boldsymbol{u}^n$ at the server using the overlay network. In order to obtain a speed up, instead of transmitting entire updates over *links*, clients may split their updates into M *chunks*, where each chunk is a vector of dimension $\dim/M$, and transmit chunks over the links in the network. Denoting the division of Client n's update into chunks as, $\boldsymbol{u}^n = (\boldsymbol{u}^n(m))_{m=1}^{M}$, the server computes *chunk-aggregates* $\boldsymbol{u}(m) \triangleq \frac{1}{N-1}\sum_{n=2}^{N} \boldsymbol{u}^n(m)$ in order to obtain the aggregate $\boldsymbol{u} = (\boldsymbol{u}(m))_{m=1}^{M}$.

At any point in time, a node that is not busy in a communication may communicate with at most one other node (which is also not busy) by setting up a link, and be either receiving data or transmitting data, but not both. The *delay* on a link is the time required to send a chunk across it. The link delays may be constant and equal for all links in the network, which we call the *synchronous setting*. Or, the link delays may be random, which we call the *asynchronous setting*. The communication model is further elaborated in Section 2.2.

Some transmissions across links may *fail*. This may happen because the link delay is too high and the protocol gives up, or if the node shuts down due to exogenous reasons. Since we employ overlay network aggregation, a Client n failing to transmit on a link may lead to more than just its update being lost as illustrated with the failure of the uplink from Client 4 in Fig. 1. In order



	N^1	N^2	N^3	N^4	N^5
Slot 1	5,1,1, rx	4,1,1,rx	∅	2,1,1,tx	1,1,1,tx
Slot 2	∅	5,2,1,tx	4,2,1,tx	3,2,1,tx	2,2,1,rx
Slot 3	3,1,1,rx	∅	1,1,1,tx	5,2,1,rx	4,2,1,tx
Slot 4	2,1,1,rx	1,1,1,tx	∅	∅	∅
Slot 5	4,2,1,rx	∅	∅	1,2,1,tx	∅

(a) Aggregation Trees (b) Aggregation Schedule (c) Node Schedules

Fig. 2. Aggregation Trees, Schedule and Node Schedules for a system with $N = 5$ nodes, $M = 2$ chunks and $K = 1$ replicas per chunk. The node schedule figure depicts a snapshot in time of the progress of nodes in an asynchronous implementation. Nodes 2 and 5 are communicating in their virtual slots 2, Nodes 1 and 3 are communicating in their virtual slots 3 and Node 4 is waiting for Node 5 in its virtual slot 3.

to protect against *failures*, we introduce redundancy by creating K replicas of each chunk and aggregating them on different trees as explained next.

2.1 Aggregation Trees

Corresponding to each chunk-replica (m, k) there is a directed rooted tree $\mathcal{T}(m, k)$, over the N nodes, with the server (Node 1) as the root, such that there exists a unique path from every client to the server. The *children* of Node n in $\mathcal{T}(m, k)$ are the set of all nodes that have a directed link to n in $\mathcal{T}(m, k)$. The *parent* of Node n in $\mathcal{T}(m, k)$ is the unique node to which n has a directed link in $\mathcal{T}(m, k)$. A node is a *leaf node* in $\mathcal{T}(m, k)$ if it has no children.

Aggregation of chunk-replica (m, k) happens as follows. A leaf node n_l sets it *message* as $v^{n_l}(m, k) \leftarrow u^{n_l}(m)$ and sends it to its parent. The parent receives the message if the link doesn't fail. For every Client n that is not a leaf node, let $C^n(m, k)$ denote the set of children in $\mathcal{T}(m, k)$ from which it has successfully received messages. Then, Client n collects messages from all these children and computes its message as, $v^n(m, k) \leftarrow u^n(m) + \sum_{n_c \in C^n(m,k)} v^{n_c}(m, k)$. Similarly, denoting by $C^1(m, k)$ the set of the server's children from which it has successfully received messages, it sets its message as $v^1(m, k) \leftarrow \sum_{n_c \in C^1(m,k)} v^{n_c}(m, k)$. If there were no failures in $\mathcal{T}(m, k)$, then the chunk-aggregate may be computed by the server as $v^1(m, k)/(N - 1)$. In case of failures, discussion on estimating the aggregate using $v^1(m, k)$'s is delayed to Section 4. An example set of aggregation trees is shown in Fig. 2a.

2.2 Communication Model and Aggregation Schedule

In this section we consider the creation of a schedule for message transfers over a collection of aggregation trees $\{\mathcal{T}(m, k)\}_{m\in[M],k\in[K]}$ that respects the constraints of communication on the overlay network. The design of these schedules is considered under the synchronous and asynchronous settings.

2.2.1 Synchronous Setting: For simplicity and ease of analysis, we first describe the aggregation schedule in the *Synchronous* setting. In the Synchronous setting, the delay of chunk transfers over all links is fixed to a constant. In particular, for any nodes n_1 and n_2, the time required to send a message (corresponding to 1 chunk-replica) from n_1 to n_2 is,

$$d_{\text{sync}}(M) = \frac{F}{M} \cdot \frac{1}{c_{\text{prop}}} + \frac{1}{c_f}, \tag{1}$$

where F is the size of the update, M is the number of chunks, c_{prop} is the transfer speed that captures the delay that is proportional to the size of the message, and c_f accounts for a fixed delay such as the time required to set up the link.

Time is split into slots, where each slot has a duration of $d_{\text{sync}}(M)$. The *aggregation schedule*, denoted by $\mathcal{S}$, is a sequence of sets where $\mathcal{S}(\text{s}) = \left\{\left(n^{\text{s}}_{i,\text{tx}}, n^{\text{s}}_{i,\text{rx}}, m^{\text{s}}_i, k^{\text{s}}_i\right)\right\}_{i=1}^{I^{\text{s}}}$ describes the set of communications that happen in slot s. I^{s} is the total number of communications that happen in slot s, $n^{\text{s}}_{i,tx}$ and $n^{\text{s}}_{i,rx}$ are the transmitter and receiver, respectively, of the i^{th} communication, and m^{s}_i and k^{s}_i identify the chunk-replica corresponding to the message sent from $n^{\text{s}}_{i,tx}$ to $n^{\text{s}}_{i,rx}$. Slot indices run from 1 to S, where S denotes the length of the schedule. The *aggregation delay* in this setting is, $T_{\text{sync}}(\mathcal{S}) = S \cdot d_{\text{sync}}(M)$. An example aggregation schedule is shown in Fig. 2b.

The aggregation schedule has to adhere to the constraints set by the communication model. Specifically, in any slot s, a node may participate in at most one communication. Further, for every chunk-replica (m, k) that is a part of the schedule, the schedule induces an aggregation tree $\mathcal{T}(m, k)$ such that, a directed link (n_1, n_2) exists in $\mathcal{T}(m, k)$ if and only if (n_1, n_2, m, k) exists in $\mathcal{S}(\text{s})$ in exactly one slot s. Additionally, a schedule $\mathcal{S}$ has to respect the *precedence constraint* set by the tree $\mathcal{T}(m, k)$ i.e., the transmission of chunk-replica (m, k) from n_1 may only be scheduled after all links from the children of n_1 in $\mathcal{T}(m, k)$ have been scheduled.

The schedule $\mathcal{S}$ can also be conceptualized as a set of *node-schedules*, where each node n has its own node-schedule denoted as $\mathcal{N}^n$. $\mathcal{N}^n$ is a list of length S, where,

$$\mathcal{N}^n(\text{s}) = \begin{cases} \emptyset, & \text{if } n \text{ doesn't participate in slot s,} \\ (n_{\text{rx}}, m, k, \text{tx}), & \text{if } (n, n_{\text{rx}}, m, k) \in \mathcal{S}(\text{s}), \\ (n_{\text{tx}}, m, k, \text{rx}), & \text{if } (n_{\text{tx}}, n, m, k) \in \mathcal{S}(\text{s}). \end{cases} \tag{2}$$

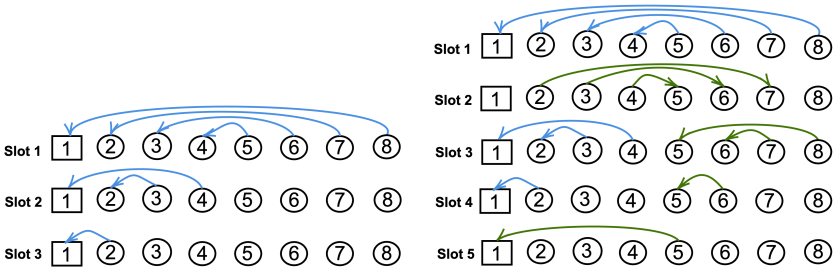
An example of the set of node schedules corresponding to the schedule in Fig. 2b is shown in Fig. 2c.

2.2.2 Asynchronous Setting: In this setting, communications no longer happen in a sequence of synchronized slots. Instead, two nodes not currently participating in any communication may set up a link and transfer a chunk-replica from one to the other. Then, the transfer delay on the link is i.i.d., distributed according to CDF $F_{D(M)}$. The mean, $\mu_{D(M)}$ of the link delay distribution is equal to $d_{\text{sync}}(M)$.

In this setting, we implement an asynchronous version of schedule $\mathcal{S}$ where communications are scheduled when the transmitter and receiver nodes are available while respecting the precedence constraints given by $\mathcal{S}$. Specifically, each node n follows its node schedule $\mathcal{N}^n$ in a sequence of virtual slots. At the start, the node is in its virtual slot 1 and it contacts its partner node in $\mathcal{N}^n(1)$ and begins the transfer of the appropriate chunk-replica. Once the transfer is complete, it moves onto its next virtual slot. If $\mathcal{N}^n(1) = \emptyset$, it immediately moves onto its next virtual slot. Once a node has completed its first $s - 1$ virtual slots, it proceeds to its s^{th} virtual slot. If $\mathcal{N}^n(s) = \emptyset$, it immediately proceeds to its $s + 1^{\text{th}}$ virtual slot. Otherwise, it checks if its partner node in $\mathcal{N}^n(s)$ has also completed its first $s - 1$ virtual slots. If it has, then the nodes set up a link and transfer a chunk-replica according to $\mathcal{N}^n(s)$. If it has not, then node n waits until it partner node completes is first $s - 1$ virtual slots. See Fig. 2c for an example.

Since the asynchronous implementation respects the precedence constraints of all the node schedules, the server ends up aggregating the same aggregate as it would in the Synchronous setting.

The aggregation delay in the Asynchronous setting, denoted $T_{\text{async}}(\mathcal{S})$, is the time at which all nodes have completed their respective schedules.



(a) $\mathcal{S}_1$ produced by OptSched(8, {1, 2}, 1) (b) $\mathcal{S}_2$ produced by OptSched(8, {1, 2}, 1)

Fig. 3. Chunk-replica 1's message transfers are shown in blue, and 2's are shown in green.

3 OptSched: A SCHEDULE FOR OVERLAY AGGREGATION WITH CHUNKING

In this section, we propose a protocol to construct an aggregation schedule and later prove that the schedule is optimal in terms of minimizing the length of the schedule. For ease of exposition, we present a version of our protocol OptSched for the special case when the number of nodes N is a power of 2 here. The complete protocol for a general N may be found in Appendix C.

Looking ahead, when running our aggregation and estimation protocol OptAgg under failures, we may need to reattempt aggregation of a subset of chunks, say, $\mathcal{M} \subseteq [M]$. Therefore, OptSched accepts as inputs the number of nodes N, a set of chunks $\mathcal{M}$, and the number of replicas per chunk K, and outputs a schedule $\mathcal{S}_{|\mathcal{M}|K}$ that aggregates the $|\mathcal{M}|\,K$ chunk-replicas.

The intuition underlying how OptSched constructs schedules may be gleaned from an example for $N = 8$, $\mathcal{M} = \{1, 2\}$ and $K = 1$ shown in Fig. 3. Let the 2 chunk-replicas be labelled 1 and 2. It first creates a schedule $\mathcal{S}_1$ for chunk-replica 1 where at each slot, the maximum number of communications are scheduled amongst the nodes that haven't already transmitted the chunk-replica, ref., Fig. 3a. Schedule $\mathcal{S}_1$ has two phases, first is the *head* phase which ends at the last slot where all N nodes are busy, and the rest of the slots are in the *tail* phase. Observe that Nodes 5, 6, 7 and 8 are idle in the tail phase of $\mathcal{S}_1$. Schedule $\mathcal{S}_2$ for two chunk-replicas is constructed based on $\mathcal{S}_1$ as follows: it is identical to $\mathcal{S}_1$ in the head phase, and *interleaves* a slot between the head and tail phases of $\mathcal{S}_1$ where Nodes 2, 3 and 4 transmit chunk-replica 2 to Nodes 5, 6 and 7. Then, as Nodes 1, 2, 3 and 4 aggregate chunk-replica 1 in the tail, Nodes 5, 6, 7 and 8 simultaneously aggregate chunk-replica 2. After the chunk-replica 1's aggregate has been received at the server in slot 4, chunk-replica 2's aggregate present at Node 5 is transmitted to the server in slot 5, ref. Fig. 3b. $\mathcal{S}_2$ takes 2 extra slots compared to $\mathcal{S}_1$. In general, OptSched constructs a schedule $\mathcal{S}_k$ for k chunk-replicas from a schedule $\mathcal{S}_{k-1}$ for $k-1$ chunk-replicas by identifying the head and tail phases of $\mathcal{S}_{k-1}$, and interleaving a slot and scheduling simultaneous aggregation in the tail phase.

A formal description is presented in Algorithm 1. OptSched creates a sequence of schedules $\mathcal{S}_1, \ldots, \mathcal{S}_{|\mathcal{M}|K}$ iteratively, where $\mathcal{S}_{k'}$ is a schedule to aggregate k' of the $|\mathcal{M}|\,K$ chunk-replicas. A bijective function ϕ is used to map chunk-replica index pairs, (m, k) to a serial index k' (see Line 1).

In Lines 3 to 7, the schedule $\mathcal{S}_1$ is created. Here, as described above, in each slot, the maximum number of links are formed between nodes that haven't already transmitted $\phi^{-1}(1)$ until all client nodes have transmitted once.

For $k' > 1$, $\mathcal{S}_{k'}$ is constructed from $\mathcal{S}_{k'-1}$. This operation involves four steps: *Retain*, *Interleave*, *Merge*, and *Final Aggregation*. In the Retain step, which is for the first l slots of the head phase (l being defined in Line 10), $\mathcal{S}_{k'}$ follows the same schedule as $\mathcal{S}_{k'-1}$.

Algorithm 1: OptSched $(N, \mathcal{M}, K)$ where N is a power of 2.

Input : Number of Nodes: N, (labelled 1 to N)
Chunk Indices: $\mathcal{M}$,
Replicas Per Chunk: K,

Output : Schedule $\mathcal{S}_{|\mathcal{M}|K}$ to compute $\sum_{n=2}^{N} \boldsymbol{u}^n(m,k)$ at node 1 for all $m \in \mathcal{M}, k \in [K]$.

1 Compute a bijective function, $\phi : \mathcal{M} \bigtimes [K] \rightarrow [|\mathcal{M}|\, K]$;

// Initialize Schedule

2 $N' = N$, s = 1 ;

3 **while** $N' > 1$ **do**

4 $\quad \mathcal{S}_1(\mathrm{s}) = \left\{(N' + 1 - j, j, \phi^{-1}(1)) : j = 1 \ldots N'/2\right\}$;

5 $\quad N' = N'/2$;

6 $\quad$ s = s +1 ;

7 **end**

8 **for** $k' = 2$ **to** $|\mathcal{M}|\, K$ **do**

9 $\quad \mathrm{S}_{k'-1} = \mathrm{len}(\mathcal{S}_{k'-1})$;

10 $\quad l = \mathrm{S}_{k'-1} - \log N + 1$;

$\quad$ // Retain

11 $\quad \mathcal{S}_{k'}(\mathrm{s}) = \mathcal{S}_{k'-1}(\mathrm{s}), \quad 1 \leq \mathrm{s} \leq l$;

$\quad$ // Interleave

12 $\quad \mathcal{N}_{\mathrm{tx}} = (n : n \geq 2, \text{client } n \text{ participates in slot } l + 1 \text{ or later of } \mathcal{S}_{k'-1})$; // $|\mathcal{N}_{\mathrm{tx}}| = N/2 - 1$

13 $\quad \mathcal{N}_{\mathrm{rx}} = (n : n \geq 2, \text{client } n \text{ does not participate in slot } l + 1 \text{ or later of } \mathcal{S}_{k'-1})$;

14 $\quad \mathcal{S}_{k'}(l+1) = \left\{\left(\mathcal{N}_{\mathrm{tx}}(j), \mathcal{N}_{\mathrm{rx}}(j), \phi^{-1}(k')\right) : j = 1 \ldots |\mathcal{N}_{\mathrm{tx}}|\right\}$;

$\quad$ // Merge

15 $\quad \mathcal{N}^0 = [2, N] \setminus \mathcal{N}_{\mathrm{tx}}$; // nodes yet to transmit $\phi^{-1}(k')$

16 $\quad$ **for** s = $l + 2$ **to** $\mathrm{S}_{k'-1}$ +1 **do**

17 $\quad\quad \mathcal{N}_{\mathrm{avail}} = \mathcal{N}^0 \setminus \{\text{nodes participating in } \mathcal{S}_{k'-1}(\mathrm{s}-1)\}$;

18 $\quad\quad \mathcal{S}^0 = \{\left(\mathcal{N}_{\mathrm{avail}}(|\mathcal{N}_{\mathrm{avail}}| + 1 - j), \mathcal{N}_{\mathrm{avail}}(j), \phi^{-1}(k')\right) : j = 1 \ldots \lfloor |\mathcal{N}_{\mathrm{avail}}|/2 \rfloor\}$;

19 $\quad\quad \mathcal{S}_{k'}(\mathrm{s}) = \mathcal{S}_{k'-1}(\mathrm{s}-1) \cup \mathcal{S}^0$;

20 $\quad\quad \mathcal{N}^0 = \mathcal{N}^0 \setminus \{\text{ transmitters in } \mathcal{S}^0\}$;

21 $\quad$ **end**

$\quad$ // Final Aggregation

22 $\quad \mathcal{S}_{k'}(\mathrm{S}_{k'-1} + 2) = \{(\mathcal{N}_{\mathrm{avail}}(1), 1, \phi^{-1}(k'))\}$;

23 **end**

Before describing the Interleave step, we observe that in the tail phase of $\mathcal{S}_{k'-1}$, there will be a set of nodes that have transmitted all of the first $k' - 1$ chunk-replicas ordered according to ϕ^{-1} and, thus, do not participate anymore in $\mathcal{S}_{k'-1}$. So, looking forward, in the Merge step, the aim is to aggregate the new chunk-replica in these slots with these nodes. To that end, in the Interleave step, a set of nodes that are active in the tail pase of $\mathcal{S}_{k'-1}$ are identified (see Line 12), and in Line 14 a new slot is interleaved where these active nodes transmit the new chunk-replica $\phi^{-1}(k')$ to the rest of the nodes. As described earlier, in the Merge step, $\mathcal{S}_{k'}$ continues to aggregate the first $k' - 1$ chunk-replicas as was performed in the tail of $\mathcal{S}_{k'-1}$, while simultaneously aggregating

chunk-replica $\phi^{-1}(k')$ amongst the nodes that are not participating in the aggregation of the first $k'-1$ chunk-replicas.

As will be shown in the proof of Lemma 2, by the end of the Merge step, the new chunk-replica has been aggregated at a single client. So, in the Final Aggregation step, this client transmits the aggregated chunk-replica to the server (see Line 22), which completes the schedule, thus aggregating all the k' chunk-replicas.

Remark 1. *In Lines 4, 12 and 13, ordered sets of nodes are formed, although the ordering of these nodes may be arbitrary. In the sequel (see Section 5.2) we will randomize such ordering to provide additional robustness to link delay and failure heterogeneity.*

Next, we analyze the aggregation delay of OptSched in the Synchronous and Asynchronous settings.

3.1 Optimality of OptSched in the Synchronous Setting

In the Synchronous setting, for any schedule $\mathcal{S}$ of length S that aggregates chunks $\mathcal{M} \subseteq [M]$ and K replicas per chunk over N nodes, the aggregation delay is simply, $T_{\text{sync}}(\mathcal{S}) = S \cdot d_{\text{sync}}(M)$. Therefore, to prove the optimality of the aggregation delay of the schedule produced by OptSched, one needs to prove that the length of the schedule is optimal. The following Lemma provides a lower bound on the length of any schedule $\mathcal{S}$.

Lemma 1. *Let $\mathcal{S}$ be a schedule that aggregates a set $\mathcal{M} \subseteq [M]$ of chunks, with K replicas each. Then, the length S of the schedule is lower bounded as,*

$$S \geq \lceil \log N \rceil + \frac{2(|\mathcal{M}|\,K-1)(N-1)}{N}.$$

We provide a proof sketch here. There are a total of $|\mathcal{M}|\,K$ chunk replicas, and each one requires $N-1$ communications to be aggregated since each client has to transmit each chunk replica exactly once. At most $N/2$ transmissions can be realized in each slot. Therefore, a lower bound on the length of the schedule is $2\,|\mathcal{M}|\,K(N-1)/N$. However, a tighter lower bound can be obtained by recognizing that to complete aggregation, only one communication may happen in the last slot where a client sends the last message to the server. Similarly, in the penultimate slot, at most 2 communications may happen where two other nodes transmit to the two nodes that communicate in the last slot. Extrapolating, for $0 \leq t \leq \lfloor \log N \rfloor - 1$, at most 2^t communications may happen in the $S-t^{\text{th}}$ slot. Up to $N/2$ communications may happen in all previous slots. Using this constraint, and the fact that $|\mathcal{M}|\,K(N-1)$ total communications need to happen, we obtain the lower bound.

The following Lemma characterizes the length of the schedule produced by OptSched.

Lemma 2. OptSched$(N, \mathcal{M}, K)$ *produces an aggregation schedule $\mathcal{S}_{|\mathcal{M}|K}$ with length $S_{|\mathcal{M}|K} = \lceil \log N \rceil + 2(|\mathcal{M}|\,K-1)$ slots.*

We provide a sketch here. The length of the schedule $\mathcal{S}_1$ is $\lceil \log N \rceil$ because at the end of each slot, the number of nodes that are yet to transmit is halved. And, when constructing $\mathcal{S}_{k'}$ from $\mathcal{S}_{k'-1}$, exactly two more slots are added (one associated with the Interleave step, and the other associated with the Final Aggregation step). Therefore, the length of the schedule $\mathcal{S}_{|\mathcal{M}|K}$ is $\lceil \log N \rceil + 2(MK-1)$. The complete proof, given in Appendix D involves showing that $\mathcal{S}_{|\mathcal{M}|K}$ indeed aggregates all the chunk replicas.

With the observation that in the Synchronous setting the aggregation delay of a schedule is simply the length of the schedule scaled by the slot duration, Lemma 1 and 2 prove that OptSched is optimal in this setting.

Theorem 1. *Consider a system in the Synchronous setting with link delays $d_{\text{sync}}(M)$. Let $\mathcal{S}$ be any schedule that aggregates chunks in $\mathcal{M} \subseteq [M]$, with associated aggregation delay $T_{\text{sync}}(\mathcal{S})$. Let $\mathcal{S}_{|\mathcal{M}|K}$ be the schedule produced by* OptSched$(N, \mathcal{M}, K)$ *with aggregation delay $T_{\text{sync}}(\mathcal{S}_{|\mathcal{M}|K})$. Then, $T_{\text{sync}}(\mathcal{S}_{|\mathcal{M}|K})/T_{\text{sync}}(\mathcal{S}) \leq 1 + O(1/N)$.*

Remark 2. *The slot duration $d_{\text{sync}}(M)$ is a decreasing function of the number of chunks M. And, the length of the schedule is an increasing function of the number of chunks. This suggests that the aggregation delay is a U-shaped function of the number of chunks. The aggregation delay is high when the files are split into a small number of chunks because then the schedule cannot exploit the parallelism possible in the link communications. On the other hand, if the file is split into too many chunks, the aggregation delay is also high because a large number of links need to be setup, and the fixed cost $1/c_f$ of setting up links adversely affects the aggregation delay. Therefore, there exists a sweet spot for the number of chunks that minimizes the aggregation delay and is the nearest integer to,*

$$\begin{aligned} M^* &= \arg\min_{M \in \mathbb{N}} \left(\lceil \log N \rceil + 2(MK - 1)\right) d_{\text{sync}}(M), \\ &= \sqrt{\frac{F c_f \left(\lceil \log N \rceil - 2\right)}{2K c_{\text{prop}}}} \end{aligned} \tag{3}$$

3.2 Near-Optimality of OptSched in the Asynchronous Setting

The aggregation delay in the Asynchronous setting, $T_{\text{async}}(\mathcal{S})$, is a more complicated function of the schedule since communication doesn't happen in a sequence of synchronized slots. As such, we simply state the lower bound, upper bound and near-optimality of the aggregation delay of OptSched here, and postpone the analysis required to obtain these results to Appendix A.

The following Lemma gives a lower bound on the aggregation delay of any schedule in the Asynchronous setting.

Lemma 3. *Consider a system in the Asynchronous setting with link delay CDF $F_{D(M)}$ and mean $\mu_{D(M)}$. Then, the expected aggregation delay of a schedule $\mathcal{S}$ of length S is lower bounded as,*

$$\mathbb{E}\left[T_{\text{async}}(\mathcal{S})\right] \geq S \cdot \mu_{D(M)}.$$

Recall that the aggregation delay of schedule $\mathcal{S}$ in the Synchronous setting is $T_{\text{sync}}(\mathcal{S}) = S \cdot d_{\text{sync}}(M)$. Therefore, Lemma 3 says that the expected aggregation delay of a schedule in the Asynchronous setting is no better than the aggregation delay in the Synchronous setting when the mean link delay remains the same.

To develop an upper bound on the aggregation delay in the Asynchronous setting, we require that the link delay CDF $F_{D(M)}$ has a light tail. In particular, we require that it have a log moment generating function (logMGF) denoted as, $\Lambda_{D(M)}(\theta) \triangleq \ln \mathbb{E}\left[e^{\theta D(M)}\right]$, where $D(M) \sim F_{D(M)}$. The upper bound will make use of the distribution's *right-tail Cramér function*,

$$\Lambda^*_{D(M)}(\alpha) = \begin{cases} 0 & \text{, if } \alpha < \mathbb{E}[D(M)], \\ \sup_{\theta \geq 0}[\theta\alpha - \Lambda_{D(M)}(\theta)] & \text{, if } \alpha \geq \mathbb{E}[D(M)]. \end{cases}$$

In the domain where it is positive and finite, $\Lambda^*_{D(M)}(\alpha)$ is strictly increasing in α.

The following Lemma characterizes an upper bound on the aggregation delay of any schedule $\mathcal{S}$ in the Asynchronous setting when the link delay distribution has a light tail.

Lemma 4. *Consider a system in the Asynchronous setting with link delay CDF $F_{D(M)}$ that has a logMGF $\Lambda_{D(M)}(\theta)$ and a right-tailed Cramér function $\Lambda^*_{D(M)}(\alpha)$. Let $\alpha^*(M) > 0$ be the unique solution*

to the equation, $\Lambda^*_{D(M)}(\alpha^*(M)) = \ln 2$. *Then, for a schedule* $\mathcal{S}$ *of length* S *and any* $\alpha > \alpha^*(M)$, *the right tail of the aggregation delay is upper bounded as,*

$$P\left(T_{\text{async}}(\mathcal{S}) \geq \alpha\, S\right) \leq e^{-S(\Lambda^*_{D(M)}(\alpha) - \ln 2)}.$$

Further, the expected aggregation delay is upper bounded as, $\mathbb{E}\left[T_{\text{async}}(\mathcal{S})\right] \leq \alpha^*(M)\, S + o(S)$.

Combining the lower and upper bounds on the aggregation delays of schedules in the Asynchronous setting of Lemma 3 and 4 with the bounds on the length of the schedules from Lemma 1 and 2, we get the near-optimality result for OptSched.

Theorem 2. *Consider a system in the Asynchronous setting with the link delay CDF* $F_{D(M)}$ *with mean* $\mu_{D(M)}$. *Let* $F_{D(M)}$ *have a logMGF* $\Lambda_{D(M)}(\theta)$ *and right-tailed Cramér function* $\Lambda^*_{D(M)}(\alpha)$, *and let* $\alpha^*(M)$ *be given as in Lemma 4. Let* $\mathcal{S}$ *be any schedule that aggregates* K *replicas of chunks* $\mathcal{M} \subseteq [M]$ *and* $\mathcal{S}_{|\mathcal{M}|K}$ *be the schedule produced by* OptSched$(N, \mathcal{M}, K)$. *Then,*

$$\frac{\mathbb{E}\left[T_{\text{async}}\left(\mathcal{S}_{|\mathcal{M}|K}\right)\right]}{\mathbb{E}\left[T_{\text{async}}\left(\mathcal{S}\right)\right]} \leq \frac{\alpha^*(M)}{\mu_{D(M)}} + o(S).$$

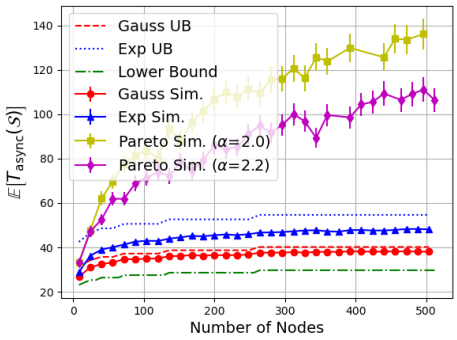
Remark 3. *To capture an intuition for why the result of Theorem 2 is surprising, observe that roughly* $N/2$ *links are active at any time in the Asynchronous setting. So, with* N *being large, there is an increased chance of there being a straggler link, i.e., a link with very high delay. A straggler link may slow down the entire schedule because other nodes may be scheduled to communicate with nodes involved in the straggler link, and therefore have to wait for the straggler link to complete. Surprisingly, Theorem 2 says that when the link delay distribution has a logMGF, no matter how large the number of nodes, the expected aggregation delay in the Asynchronous setting is at most a constant multiplicative gap away to a lower bound, with the lower bound being equal to the aggregation delay in a Synchronous setting with the same mean link delay.*

To further illustrate the phenomenon in Theorem 2 and Remark 3, we present some simulation results under 3 different link delay distribution settings. Consider a system with file size $F = 100Mb$, proportional speed $c_{\text{prop}} = 10Mbps$ and fixed delay $1/c_f = 0.1s$. We show simulation results under three link delay distributions that all have a mean $d_{\text{sync}}(M)$. They are a shifted Exponential, Gaussian distribution and a Pareto distribution with scale $\alpha > 1$ (unrelated to α^* in Theorem 2). The distributions are given in Table 1.

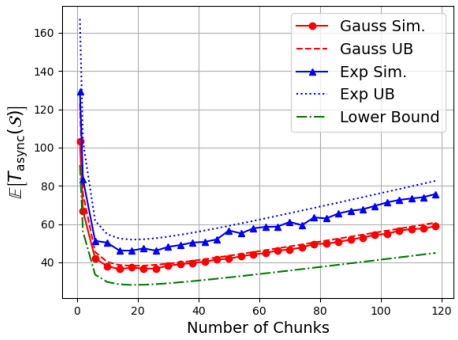
Shifted Exponential (Exp)	$f^E_{D(M)}(x) = (2/d_{\text{sync}}(M))e^{-\frac{2}{d_{\text{sync}}(M)}(x - d_{\text{sync}}(M)/2)}, \quad x \geq \frac{d_{\text{sync}}(M)}{2}$
Gaussian (Gauss)	$f^G_{D(M)} = \mathcal{N}(d_{\text{sync}}(M), (0.3d_{\text{sync}}(M))^2)$
Pareto(α)	$f^P_{D(M)}(x) = \frac{\alpha}{x^{\alpha+1}}\left(\frac{(1-\alpha)d_{\text{sync}}(M)}{\alpha}\right)^{\alpha}, \quad x \geq \frac{(1-\alpha)d_{\text{sync}}(M)}{\alpha}.$

Table 1. Link Delay Distributions considered in Simulations

Fig. 4a shows the behavior of the mean aggregation delay as the number of nodes increases for the three link delay distributions in the case where the number of chunks is 10, and the number of replicas is 1. Since the shifted Exponential and Gaussian distributions have a logMGF, their expected aggregation delay is bounded above by a distribution-dependent upper bound $\alpha^*(M)S$, where the α^*'s are $2.02d_{\text{sync}}(M)$ and $1.49d_{\text{sync}}(M)$ respectively, and S is the schedule length. Crucially, even as the number of nodes increases, their aggregation delays under OptSched are within 2.02 and 1.49 times of the lower bound given in Lemma 3. However, when the tail of the link delay



(a) Simulation of variation of aggregation delay as a function of number of nodes in the Asynchronous setting under different link delay distributions. 95% confidence intervals are shown for the Pareto distributions.



(b) Simulation of variation of aggregation delay as a function of number of chunks in the Asynchronous setting under different link delay distributions, along with the theoretical lower bound and upper bounds.

Fig. 4

distribution is heavy, such as in the Pareto distribution, this ratio between the aggregation delay and the lower bound grows with the number of nodes as can be seen in Fig. 4a, with the heavier tailed Pareto(2.0) having a higher aggregation delay than Pareto(2.2). Still, the aggregation delays for the Pareto distributions are smaller compared to direct aggregation's aggregation delay of $11N$ seconds for these system settings. And, compared to a state of the art gossip aggregation algorithm[22] that is described further in Section 5, for $N = 100$ nodes, we observe a >8x speedup with shifted Exponential and Gaussian distributed link delays, and >5x speedup with Pareto delays. In Section 5.3 we suggest a remedy of employing link delay thresholds to reduce the aggregation delays of heavy-tailed distributions further.

In Fig. 4b we consider a system with $N = 500$ nodes, $K = 1$ replica-per-chunk, and the shifted Exponential and Gaussian distributions for the link delay. The plot shows the variation of the aggregation delay of the schedule produced by OptSched as a function of the number of chunks. The plot highlights the performance gain obtained from chunking as opposed to using a single aggregation tree (i.e., $M = 1$). We observe that all the simulation curves, upper and lower bounds have a U-shape for the same reason as in the Synchronous setting. Notably, the simulation suggests that the location of the minimum of the lower bound curve is in agreement with the that of minimum of the simulation curves for both distributions. Therefore, since the number of chunks M is a design parameter, this suggests that one can choose the optimal M according to (3) in the asynchronous setting as well. In the example in Fig. 4b, $M^* = 18$ optimizes the aggregation delay.

Remark 4. *Although we consider the example of averaging updates in this work,* OptSched *may be used in any setting where the aggregation is* decomposable *[18]. Here, aggregation being decomposable means that the aggregate may be obtained by first performing aggregation within a partition of the clients, and then aggregating over the partition aggregates. Moreover, when the bitsize of all the updates, messages, and aggregates are the same, the (near) optimality results of Theorem 1 and 2 continue to hold. Examples of decomposable and size-preserving aggregations include summation, averaging, product, min, and max.*

4 AGGREGATION AND ESTIMATION UNDER FAILURES

In this section, we propose a protocol to estimate aggregates at the server when there are failures. Transmissions across some links may fail because the delay on the link was too high and the protocol gave up or if the node shut down due to exogenous reasons.

In overlay aggregation, since clients help aggregate each other's messages, a failure in an aggregation tree may lead to more than just the corresponding node's message not making it to the server. Indeed, the messages of the nodes below it would also not make it. So, when a node's message does not make it to the server in a particular aggregation tree, we call it a *node drop* for that aggregation tree. See Fig. 5 for an example of failure and drops with and without overlay aggregation.

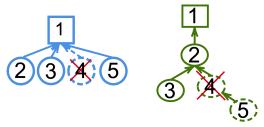


Fig. 5. The figure on the left is an example of direct aggregation. Here, node 4 failed, and it is also the only node that dropped. In the figure on the right, nodes employ overlay network aggregation. Here, node 4 failed, however, both nodes 4 and 5 dropped.

Therefore, in the presence of failures, the server may not receive the true aggregate, leading the server to make an estimate. We focus on estimates of the following kind.

Definition 1. *$\hat{\boldsymbol{u}}$ is an estimate for the aggregate $\boldsymbol{u} = \frac{1}{N-1}\sum_{n=2}^{N}\boldsymbol{u}^n$ with* linearly bounded noise *if,*
(1) it is unbiased, $\mathbb{E}[\hat{\boldsymbol{u}}|\,\boldsymbol{u}] = \boldsymbol{u}$, and,
(2) has MSE bounded as,

$$\mathbb{E}\left[\|\hat{\boldsymbol{u}}-\boldsymbol{u}\|_2^2\,\middle|\,(\boldsymbol{u}^n)_{n=2}^N\right] \le \frac{\sigma^2}{(N-1)^2}\sum_{n=2}^{N}\|\boldsymbol{u}^n\|_2^2,$$

where σ^2 is its normalized Mean-Squared-Error (MSE).

4.1 OptAgg: An Aggregation and Estimation Protocol under Failures

Here we present our aggregation and estimation protocol, OptAgg. Recall that the aggregation schedule produces an aggregation tree $\mathcal{T}(m,k)$ for each chunk-replica (m,k). Let $\mathcal{A}(m,k)$ denote the set of nodes that do not drop in the tree $\mathcal{T}(m,k)$. In the example on the right of Fig. 5, $\mathcal{A}(m,k) = \{2,3\}$, assuming that it was the aggregation tree for chunk replica (m,k). This means that the server receives the message, $\boldsymbol{v}^1(m,k) = \sum_{n\in\mathcal{A}(m,k)}\boldsymbol{u}^n(m)$ corresponding to chunk-replica (m,k). In OptAgg, the server needs to know the sizes of $\mathcal{A}(m,k)$ which may be easily obtained using the following method. Every client appends their chunk with a coordinate with entry 1. This coordinate when summed up at the server will result in the value of $|\mathcal{A}(m,k)|$, the size of $\mathcal{A}(m,k)$. OptAgg uses the value of the fraction of nodes that do not drop,

$$A(m,k) \triangleq \frac{|\mathcal{A}(m,k)|}{N-1}.$$

OptAgg computes an estimate that minimizes an upper bound on the MSE based solely on knowledge of the fraction of clients (as opposed to their identities) that successfully contributed to the aggregate, and reattempts aggregation for those that don't meet a specified MSE constraint. Intuitively, the quality of the partially received chunk-replica is higher if a higher number of nodes

Algorithm 2: OptAgg(N, M, K, σ^2_{th})

Input : Number of nodes: N,
Number of chunks: M,
Number of replicas per chunk: K,
MSE Threshold: $\sigma^2_{th} > 0$.
Output : Aggregate Estimate: $\hat{\boldsymbol{u}}$.

$\mathfrak{R} = \emptyset$; `// set of successfully received chunks`
while $|\mathfrak{R}| < M$ **do**
  Run OptSched($N, [M] \setminus \mathfrak{R}, K$) to compute schedule to aggregate chunks not in $\mathfrak{R}$;
  `// Receive chunk-replica aggregates:` $\left\{\boldsymbol{v}^1(m,k) : m \in [M] \setminus \mathfrak{R}, k \in [K]\right\}$
  `// Compute fraction of clients not dropped in aggregation tree:` $\{A(m,k) : m \in [M] \setminus \mathfrak{R}, k \in [K]\}$
  for $m \in [M] \setminus \mathfrak{R}$ **do**
    $\bar{\sigma}^2(m) = \frac{2}{\sum_{k=1}^{K} \frac{A(m,k)}{1-A(m,k)}}$;
    if $\bar{\sigma}^2(m) \leq \sigma^2_{th}$ **then**
      $\beta^*(m,k) = \frac{\frac{A(m,k)}{1-A(m,k)}}{\sum_{j=1}^{K} \frac{A(m,j)}{1-A(m,j)}}, \quad \forall k \in [K]$;
      $\hat{\boldsymbol{u}}(m) = \frac{1}{N-1} \sum_{k=1}^{K} \frac{\beta^*(m,k)}{A(m,k)} \boldsymbol{v}^1(m,k)$;
      $\mathfrak{R} = \mathfrak{R} \cup \{m\}$;
  end
  $\hat{\boldsymbol{u}} = (\hat{\boldsymbol{u}}(m))_{m=1}^{M}$
end

contributed to its aggregate received for a given tree. Our estimate takes a weighted average of such aggregates for each chunk, where the weight is higher when the number of nodes contributing, $A(m,k)$, is high. Our results show that an optimal weighting should be non-linear, i.e., be proportional to $A(m,k)/(1-A(m,k))$, which is also increasing in $A(m,k)$. OptAgg is explained in more detail next.

In addition to N, M and K, OptAgg accepts another input σ^2_{th} which is a bound on the normalized MSE of the aggregate estimate. OptAgg proceeds as follows. First, it implements the schedule produced by OptSched to aggregate all the M chunks with K replicas each. Then, for each chunk m, denoting,

$$\bar{\sigma}^2(m) \triangleq \frac{2}{\sum_{k=1}^{K} \frac{A(m,k)}{1-A(m,k)}},$$

it considers the chunk successfully received if $\bar{\sigma}^2(m) \leq \sigma^2_{th}$. We call this the *chunk-acceptance condition*, where chunks of higher quality as determined by the magnitude of $A(m,k)$ are accepted. If a chunk m is successfully received, then it computes a coefficient for the chunk's k^{th} replica as,

$$\beta^*(m,k) = \frac{\frac{A(m,k)}{1-A(m,k)}}{\sum_{j=1}^{K} \frac{A(m,j)}{1-A(m,j)}},$$

and obtains an estimate,

$$\hat{\boldsymbol{u}}(m) = \frac{1}{N-1} \sum_{k=1}^{K} \beta^*(m,k) \frac{\boldsymbol{v}^1(m,k)}{A(m,k)}.$$

Note that if $A(m,k')=1$ then, $\boldsymbol{v}^1(m,k')$ is the true aggregate of chunk m, and therefore, all partially received chunks may be ignored. In this case, we adopt the convention that $\beta^*(m,k')=1/i$ where i is the number of chunk replicas with $A(m,k')=1$, and set $\beta^*(m,k)$ to 0 for all chunk-replicas with $A(m,k)<1$. This behaviour would not be captured if, say, $\beta^*(m,k)\propto A(m,k)$.

For chunks that were not accepted, OptAgg re-attempts another aggregation. The process continues until all chunks have been successfully received. The complete protocol is described in Algorithm 2.

4.2 Analysis of OptAgg in the Independent Link Failure Model

We analyze OptAgg in the setting where each link in an aggregation tree $\mathcal{T}(m,k)$ fails independently with a probability p of every other link in all the trees $\{\mathcal{T}(m',k')\}_{m'\in\mathfrak{R},k'\in[K]}$. Particularly, even if a link between the same two nodes n_1 and n_2 exists in two different trees $\mathcal{T}(m_1,k_1)$ and $\mathcal{T}(m_2,k_2)$, they fail independently of each other. This models events where a link between two nodes failed to be setup or the transfer across the link did not complete in time due to random exogenous events in the network.

We compute a bound on the MSE of the estimate $\hat{\boldsymbol{u}}$ as a function of the fraction of clients dropped in the aggregation trees. To that end, for a chunk m, define $\mathcal{F}_{\text{drop}}(m)$ as the sigma-algebra of the fraction of clients dropped in the corresponding K aggregation trees in the attempt in which chunk m was successfully received in OptAgg,

$$\mathcal{F}_{\text{drop}}(m)\triangleq\sigma\left(\{A(m,k):k\in[K]\}\right). \tag{4}$$

Lemma 5. *Consider chunk m, and consider an estimate,*

$$\hat{\boldsymbol{u}}_\beta(m)\triangleq\frac{1}{N-1}\sum_{n=2}^{N}\frac{\beta(m,k)}{A(m,k)}\boldsymbol{v}^1(m,k),$$

where $\beta(m,k)$'s are non-negative and $\sum_{k=1}^{K}\beta(m,k)\,\mathbb{1}_{A(m,k)>0}=1$.

Under the independent link failure model, the estimate $\hat{\boldsymbol{u}}_\beta(m)$ is an unbiased estimate of $\boldsymbol{u}(m)$, $\mathbb{E}[\hat{\boldsymbol{u}}_\beta(m)|\,\boldsymbol{u}(m)]=\boldsymbol{u}(m)$. And, its MSE is bounded as,

$$\mathbb{E}\left[\left\|\hat{\boldsymbol{u}}_\beta(m)-\boldsymbol{u}(m)\right\|_2^2\,\middle|\,(\boldsymbol{u}^n(m))_{n=2}^N,\mathcal{F}_{\text{drop}}(m)\right]\le\frac{1}{(N-1)^2}\sum_{k=1}^{K}\frac{2\beta^2(m,k)}{\frac{A(m,k)}{1-A(m,k)}}\sum_{n=2}^{N}\|\boldsymbol{u}^n(m)\|_2^2.$$

The MSE bound above is minimized for $\beta(m,k)=\beta^*(m,k)$,

$$\mathbb{E}\left[\|\hat{\boldsymbol{u}}(m)-\boldsymbol{u}(m)\|_2^2\,\middle|\,(\boldsymbol{u}^n(m))_{n=2}^N,\mathcal{F}_{\text{drop}}(m)\right]\le\frac{1}{(N-1)^2}\frac{2}{\sum_{k=1}^{K}\frac{A(m,k)}{1-A(m,k)}}\sum_{n=2}^{N}\|\boldsymbol{u}^n(m)\|_2^2.$$

That is, Lemma 5, proved in Appendix F, states that the chunk estimate $\hat{\boldsymbol{u}}(m)$ obtained using OptAgg is an estimate of $\boldsymbol{u}(m)$ with linearly bounded noise that minimizes a bound on the normalized MSE.

OptAgg depends on two parameters that realize a tradeoff between aggregation delay and normalized MSE. In order to achieve a normalized MSE of no more than σ^2, we propose setting

$$\sigma_{th}^2=\sigma^2,\quad\text{and}\quad K=\left\lceil\frac{1-(1-p)\,(1-p/2)^{\lceil\log N\rceil}}{\sigma^2(1-p)\,(1-p/2)^{\lceil\log N\rceil}}\right\rceil, \tag{5}$$

where p is the link failure probability. When p is unknown, one may choose K based on a conservative estimate for p.

With such a choice of σ_{th}^2, the chunk-acceptance condition in Line 6 of Algorithm 2 motivated by the result of Lemma 5, will ensure that the normalized MSE of the estimate is bounded by σ^2.

The choice of K is more subtle. Although a small K leads to a shorter schedule length, the chances of passing the chunk-acceptance condition may be smaller leading to many reattempts and thus high aggregation delay. Our specific choice for K proposed above is developed in Appendix B. Intuitively, as can be seen, K is roughly inversely proportional to the desired normalized MSE σ^2 and is an increasing function of the link failure probability p so that the chunk-acceptance condition may be passed with a sufficient probability.

Remark 5 (Relevance to Federated Learning). *Communication is a known bottleneck in Federated Learning systems [20] and several works in the literature have proposed using client-side lossy compression of the local gradient, $\boldsymbol{g}^n$, before they are sent to the server to aggregate [1, 3, 13, 15, 31]. A standard model used for compression is a stochastic compression operator $\boldsymbol{Q}$ which is unbiased, $\mathbb{E}\left[\boldsymbol{Q}(\boldsymbol{g}^n) | \boldsymbol{g}^n\right] = \boldsymbol{g}^n$ and has MSE bounded as, $\mathbb{E}\left[\|\boldsymbol{Q}(\boldsymbol{g}^n) - \boldsymbol{g}^n\|_2^2 \mid \boldsymbol{g}^n\right] \leq q\,\|\boldsymbol{g}^n\|_2^2$, where $q > 0$ is a compression parameter. Stochastic quantization and random sparsification are two examples [1, 3]. Then, the server obtains an estimate for the aggregate, $\boldsymbol{g} = \frac{1}{N-1}\sum_{n=2}^{N} \boldsymbol{g}^n$ as, $\boldsymbol{g}_{\boldsymbol{Q}} = \frac{1}{N-1}\sum_{n=2}^{N} \boldsymbol{Q}(\boldsymbol{g}^n)$. Since clients apply compression independently on the local gradients, the MSE on $\boldsymbol{g}_{\boldsymbol{Q}}$ can be bounded as, $\mathbb{E}\left[\left\|\boldsymbol{g}_{\boldsymbol{Q}} - \boldsymbol{g}\right\|_2^2 | (\boldsymbol{g}^n)_{n=2}^N\right] \leq \frac{q}{(N-1)^2}\sum_{n=2}^{N}\|\boldsymbol{g}^n\|_2^2$. Therefore, the aggregate of the compressed gradients may be viewed as an estimate with linearly bounded noise. Previous works have shown that, depending on the FL algorithm being used, the number of rounds of training needed to converge to a specified convergence criterion grows as an increasing function of q [1, 3, 13, 31].*

Therefore, the estimate $\hat{\boldsymbol{u}}$ produced by OptAgg *functions similarly to the aggregate of client-side compressed updates. When client-side compression with parameter q is combined with* OptAgg *under a MSE constraint σ^2, the independence between the randomness of the client-side compressors and the link failures allows us to demonstrate that the server's estimate will meet the criteria of Definition 1 with a normalized MSE of $q + \sigma^2$. This highlights the relevance of our work for an FL application.*

5 SIMULATION

In this section, we present some simulation results to verify the analysis of OptAgg from Section 4.2 in the i.i.d., link delay and i.i.d., failure setting, study the generality of OptSched and OptAgg in a setting with heterogeneous link delays and link failures, and illustrate that when link delay distributions have heavy tails, it may be useful to set a threshold on link delays to get much smaller aggregation delays while only suffering a small normalized MSE on the estimate.

For all the simulations below, we consider file size $F = 100Mb$, proportional speed $c_{\text{prop}} = 10Mbps$ and fixed delay $1/c_f = 0.1s$. The number of chunks is $M = 10$, which means $d_{\text{sync}}(M) = 1.1s$. We consider the three link delay distributions given in Table 1. All simulations have been run until the 95% confidence interval has converge to a width of 5% of the sample mean of the aggregation delay and/or the normalized MSE.

In addition to comparing to the baseline of direct aggregation, below we will also compare to the gossip-based Push-Sum algorithm for aggregation introduced in [22]. Our structured aggregation approach is quite different than theirs, thus we will make a few reasonable assumptions to make the comparison fair, in fact give Push-Sum an advantage. In their setting, nodes follow the *random-contact* model where nodes choose a receiver at random and transmit a message upon successful completion of their previous transmission. This process continues until the aggregate estimate has converged. A node may be receiving from multiple other nodes concurrently in their random-contact model. In our communication model, a node may receive from at most one node at a time. In order to compare to OptSched and OptAgg, we assumed that if a node is receiving from i other

nodes at the same time in Push-Sum, then the instantaneous rate of all the i communications is $1/(d_{\text{sync}}(M)i)$.

Note that when using Push-Sum, the server does not know exactly when the MSE of its current estimate has satisfied the desired MSE-constraint. So, in practice, one may need to run Push-Sum for a conservatively longer time to ensure that the server's average estimate has converged. To realize an aggressive comparison, we simulated a genie based stopping time for Push-Sum, i.e., stop when MSE satisfies the constraint. Therefore, Push-Sum may have higher delays than those reported in our simulations. OptAgg does not have the problem stated above because it automatically checks for a chunk-acceptance condition in order to satisfy the MSE constraint without relying on a genie knowledge of the true aggregate.

5.1 Tradeoffs in i.i.d., Link Delay Setting with Independent Failures

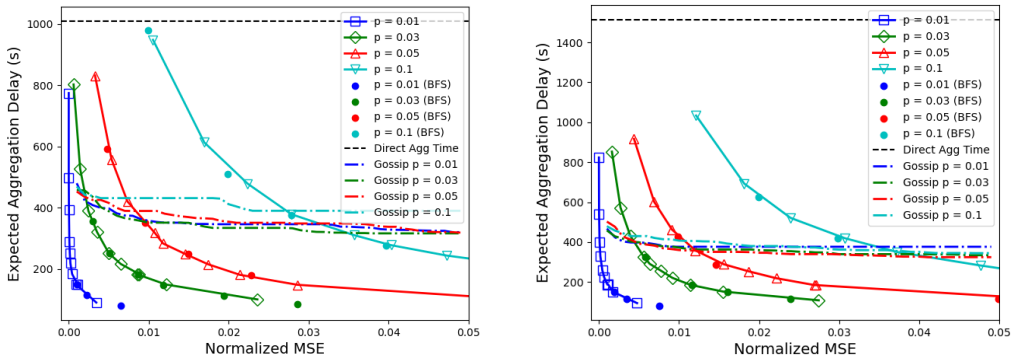
Here we consider the setting where link delays are i.i.d., distributed according to the Shifted Exponential distribution. We consider i.i.d., link failure probabilities of 0.01, 0.03, 0.05 and 0.1. We show simulation results for N=100 nodes and N=150 nodes in Fig. 6. The plots were obtained using the following procedure. Given a link failure probability p and a constraint on the normalized MSE σ^2 that the aggregate needs to satisfy, we chose σ^2_{th} and K according to (5). Then, we ran simulations to compute the expected aggregation delay and the realized normalized MSE. We verified that the realized normalized MSE is smaller than the constraint σ^2. Then we plot the expected aggregation delay vs normalized MSE. Note that normalized MSE values in the plots are the realized normalized MSE and not the normalized MSE constraints. The set of normalized MSE constraints σ^2 were uniformly chosen in the range $0.1p$ to p.

First, we observe that for all the failure probabilities considered, the expected aggregation delay is significantly lower than that of direct aggregation for the whole range of normalized MSE considered for both $N = 100$ and $N = 150$ cases. Moreover, even though individual aggregation trees are more susceptible to link failures (see Fig. 5), because the different replicas of the chunks of a node are routed along different routes in the schedule, and OptAgg aggregates and estimates information from these replicas optimally, we are able to achieve low normalized MSE while keeping the aggregation delay low as well. Therefore, we observe that OptAgg achieves a lower aggregation delay compared to the gossip algorithm down to a normalized MSE constraint $\sigma^2 \approx 0.3p$, for all the link failure probabilities p considered, even though gossip is particularly resilient to link failures. For instance, to achieve a σ^2 of 0.02 under link failure probability $p = 0.05$ with $N = 100$ nodes, OptAgg achieves a 2x speedup over gossip.

The MSE threshold in OptAgg is chosen to optimize a bound on the normalized MSE. And, the number of replicas K is chosen according to a heuristic as explained in Section B. Therefore, we also performed a brute force search (BFS) over a grid of values of σ^2_{th} and K in the following manner. Given a σ^2 constraint, we performed a grid search over a range of values of σ^2_{th} and K. Amongst the parameters that satisfied the σ^2 constraint, we chose the pair with the smallest expected aggregation delay and plotted it as a solid circle in Fig. 6. We observe that these points coincide exactly with the curve obtained using our heuristic in most cases, and are only marginally lower in a few cases. This suggests that our heuristic to choosing σ^2_{th} and K provides the *Pareto Frontier* of the achievable pairs of expected aggregation delay and normalized MSE under OptAgg. A Proof of this statement is left to future work.

5.2 Tradeoffs in the Heterogeneous Link Delay Setting with Dependent Failures

The near-optimality of the expected aggregation delay of OptSched in Theorem 2 and the MSE bound on the estimate under failures in Lemma 5 have been analyzed so far in the i.i.d., link delay



(a) N=100 (b) N=150

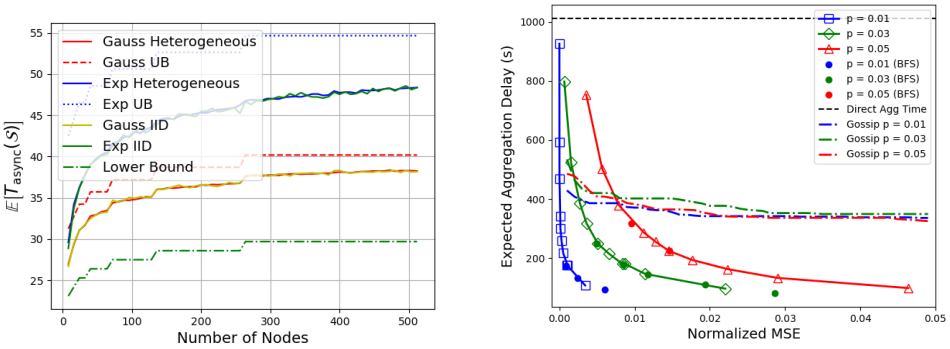
Fig. 6. Aggregation Delay and normalized MSE Tradeoff curves for OptAgg in the i.i.d., link delay with i.i.d., failures setting for $N = 100$ and $N = 150$ nodes. The solid lines show the tradeoffs by using K as in (5). The solid circles show the tradeoffs obtained by doing a brute-force search (BFS) over values of σ^2_{th} and K.

and link failure settings respectively. In this section, we study the generality of these results in heterogeneous link delay and link failure settings.

To model heterogeneous link delays, each link gets assigned a delay i.i.d., according to a distribution at time 0, and the delay for these links is then fixed for the duration of aggregation. This models a heterogeneous link delay setting because some links get assigned a high delay, and others get assigned a low delay for the entire duration of aggregation. We shall compare this to the homogeneous setting where link delays are independently resampled from the same distribution each time a schedule uses a link (i.e., the model in the analysis of the asynchronous version of OptSched in Section 3.2).

Fig. 7a shows the behavior of the expected aggregation delay of OptSched as a function of the number of nodes. Perhaps surprisingly, we observe that for both the Shifted Exponential and Gaussian distributions, the expected aggregation delay for the heterogeneous and the homogeneous settings match. Although for the heterogeneous setting link delays stay fixed across the aggregation schedule, because OptSched randomizes when and where links are included in the schedule as explained in Remark 1, the average aggregation delay behaves as if link delays were resampled at each use in the schedule. We observed the same phenomenon for the Pareto distribution as well, but we don't show it in Fig. 7a because the high aggregation delays under the Pareto distribution would hide the details of the Shifted Exponential and Gaussian plots.

Next, we consider a heterogeneous link failure setting as well. Here, the link delays are sampled heterogeneously in the same way as described above. A link failure occurs when the link delay is greater than a certain threshold d_{th}. Because the link delays themselves are fixed and heterogeneous, the link failures are fixed and heterogeneous as well. In the simulations results we present here, we consider the Pareto distribution with $\alpha = 2$ for the link delays and set the delay thresholds such that a link's failure probability is p. We consider $p = 0.01, 0.03$ and 0.05. Fig. 7b shows the expected aggregation delay and MSE tradeoffs. The curves were obtained using the same methodology as explained in Section 5.1. Here again we observe the same improvement in aggregation delay compared to direct aggregation and gossip as in the i.i.d., link delay and failure setting. Moreover, even though link failures are heterogeneous and correlated across time, by comparing against the



(a) Expected Aggregation Delay as a function of the number of nodes in the heterogeneous and i.i.d., settings.

(b) Aggregation Delay and MSE Tradeoffs of OptAgg in the heterogeneous setting with N=100 Nodes. The solid lines show the tradeoffs by using K as in (5). The solid circle show the tradeoff's obtained by doing a brute-force search (BFS) over values of σ^2_{th} and K.

Fig. 7

solid circles obtained using a brute-force search (BFS) over σ^2_{th} and K, our heuristic for choosing σ^2_{th} and K as given in (5) appear to describe the Pareto frontier for the achievable region of expected aggregation delay and normalized MSE in this setting as well. Again, the reason for the tradeoff curves in the heterogeneous link failure setting behaving similarly to those in the i.i.d., link failure setting is the randomization used by OptSched as explained above.

5.3 Delay Threshold for Heavy-Tailed Distributions

In Theorem 2, we showed that if link delay distributions are light-tailed and have a logMGF, then the expected aggregation delay of the asynchronous implementation of OptSched is near-optimal. And, in Fig. 4a we illustrated that heavy-tailed link delay distributions such as Pareto distribution may suffer a large expected aggregation delay as the number of nodes grows.

Here we propose a fix for this problem for heavy-tailed link delay distributions by using a threshold on link delays. That is, if the link delay is greater than a certain threshold d_{th}, we abandon the chunk transfer on the link and the nodes move on to their next communication in their node schedules. Since we want to compare the improvement in aggregation delay due to the delay threshold alone, we set $K = 1$, but set $\sigma^2_{th} = 8$ to reattempt aggregation for chunks in the rare events where the server receives no information from a chunk because the link from the client sending the aggregate to the server failed. The threshold d_{th} is set relatively high such that probability of link failure is low so that the normalized MSE is very low. We consider $N = 100$ nodes.

The results are shown in Fig. 8. We observe that systems with a heavy-tailed link delay distribution offer a significant improvement in aggregation delay, while only suffering a small loss in the normalized MSE. The gains are larger for the heavier-tailed distributions, as we see increasing gains with decreasing α parameter of the Pareto distribution. Moreover, the gains appear to flatten out with lower delay thresholds (corresponding to the higher MSE points in the plot). This is because a truncated Pareto distribution has a logMGF, and thus the near-optimality result of Theorem 2 negates the need to set a very low threshold.

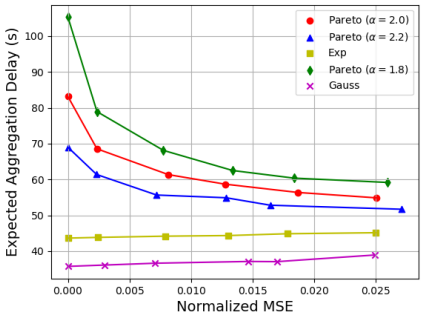


Fig. 8. Decrease in expected aggregation delay with link delay thresholding for different link delay distributions. Points with higher normalized MSE correspond to lower delay thresholds.

Lastly, we observe that there is no improvement in aggregation delay obtained by delay thresholding for the light-tailed Shifted Exponential and Gaussian distributions. In fact, there is a very slight increase in the aggregation delay as the delay threshold decreases because of reattempts made due to higher chances of not meeting the $\sigma^2_{th} = 8$ threshold. This further supports the near-optimality result of Theorem 2 for light-tailed link delay distributions.

6 CONCLUSIONS

In this paper, we have taken a new look at information aggregation, a fundamental problem in distributed systems, and one that is particularly relevant for new classes of applications such as federated learning.

We have seen that an asynchronous implementation of delay-optimal synchronous structured overlay-based aggregation schedules is robust to delay variability particularly when link delays are light-tailed. Unfortunately in networks with losses such schedules are fragile since aggregates corresponding to multiple nodes may be lost with a single failure. However by using an appropriate amount of redundancy and failure-aware estimation of the desired aggregates, one can realize trade-offs in the aggregation delay vs the normalized MSE of the estimated aggregate, giving substantial improvements over direct aggregation.

7 ACKNOWLEDGEMENTS

The authors would like to thank the reviewers and shepherd of the ACM SIGMETRICS 2025 submission of this paper. Their feedback and comments helped us immensely in improving the content and presentation of our work.

REFERENCES

[1] Dan Alistarh, Demjan Grubic, Jerry Li, Ryota Tomioka, and Milan Vojnovic. 2017. QSGD: Communication-efficient SGD via gradient quantization and encoding. *Advances in neural information processing systems* 30 (2017).

[2] Amotz Bar-Noy and Shlomo Kipnis. 1994. Broadcasting multiple messages in simultaneous send/receive systems. *Discrete Applied Mathematics* 55, 2 (1994), 95–105.

[3] Debraj Basu, Deepesh Data, Can Karakus, and Suhas Diggavi. 2019. Qsparse-local-SGD: Distributed SGD with quantization, sparsification and local computations. *Advances in Neural Information Processing Systems* 32 (2019).

[4] Beneyaz Ara Begum and Satyanarayana V Nandury. 2023. Data aggregation protocols for WSN and IoT applications–A comprehensive survey. *Journal of King Saud University-Computer and Information Sciences* 35, 2 (2023), 651–681.

[5] Yitzhak Birk, Idit Keidar, Liran Liss, Assaf Schuster, and Ran Wolff. 2006. Veracity radius: capturing the locality of distributed computations. In *Proceedings of the twenty-fifth annual ACM symposium on Principles of distributed*

computing. 102–111.
[6] Stephen Boyd, Arpita Ghosh, Balaji Prabhakar, and Devavrat Shah. 2004. Analysis and optimization of randomized gossip algorithms. In *2004 43rd IEEE Conference on Decision and Control (CDC)(IEEE Cat. No. 04CH37601)*, Vol. 5. IEEE, 5310–5315.
[7] Nicolas Broutin and Luc Devroye. 2006. Large deviations for the weighted height of an extended class of trees. *Algorithmica* 46 (2006), 271–297.
[8] Jeffrey Considine, Feifei Li, George Kollios, and John Byers. 2004. Approximate aggregation techniques for sensor databases. In *Proceedings. 20th International Conference on Data Engineering*. IEEE, 449–460.
[9] Arthur M Farley. 1980. Broadcast time in communication networks. *SIAM J. Appl. Math.* 39, 2 (1980), 385–390.
[10] Arthur M Farley, Stephen T Hedetniemi, Sandra Mitchell Mitchell, and Andrzej Proskurowski. 1979. Minimum broadcast graphs. *Discret. Math.* 25, 2 (1979), 189–193.
[11] Nadeen Gebara, Manya Ghobadi, and Paolo Costa. 2021. In-network aggregation for shared machine learning clusters. *Proceedings of Machine Learning and Systems* 3 (2021), 829–844.
[12] Indranil Gupta, Robbert Van Renesse, and Kenneth P Birman. 2001. Scalable fault-tolerant aggregation in large process groups. In *2001 International Conference on Dependable Systems and Networks*. IEEE, 433–442.
[13] Farzin Haddadpour, Mohammad Mahdi Kamani, Aryan Mokhtari, and Mehrdad Mahdavi. 2021. Federated learning with compression: Unified analysis and sharp guarantees. In *International Conference on Artificial Intelligence and Statistics*. PMLR, 2350–2358.
[14] Theodore Edward Harris et al. 1963. *The theory of branching processes*. Vol. 6. Springer Berlin.
[15] Parikshit Hegde, Gustavo de Veciana, and Aryan Mokhtari. 2023. Network adaptive federated learning: Congestion and lossy compression. In *IEEE INFOCOM 2023-IEEE Conference on Computer Communications*. IEEE, 1–10.
[16] Márk Jelasity, Alberto Montresor, and Ozalp Babaoglu. 2005. Gossip-based aggregation in large dynamic networks. *ACM Transactions on Computer Systems (TOCS)* 23, 3 (2005), 219–252.
[17] Paulo Jesus, Carlos Baquero, and Paulo Sérgio Almeida. 2009. Fault-tolerant aggregation by flow updating. In *Distributed Applications and Interoperable Systems: 9th IFIP WG 6.1 International Conference, DAIS 2009, Lisbon, Portugal, June 9-11, 2009. Proceedings 9*. Springer, 73–86.
[18] Paulo Jesus, Carlos Baquero, and Paulo Sérgio Almeida. 2014. A survey of distributed data aggregation algorithms. *IEEE Communications Surveys & Tutorials* 17, 1 (2014), 381–404.
[19] Kumar Joag-Dev and Frank Proschan. 1983. Negative Association of Random Variables with Applications. *The Annals of Statistics* 11, 1 (1983), 286–295. http://www.jstor.org/stable/2240482
[20] Peter Kairouz, H Brendan McMahan, Brendan Avent, Aurélien Bellet, Mehdi Bennis, Arjun Nitin Bhagoji, Kallista Bonawitz, Zachary Charles, Graham Cormode, Rachel Cummings, et al. 2021. Advances and open problems in federated learning. *Foundations and Trends® in Machine Learning* 14, 1–2 (2021), 1–210.
[21] Richard M Karp, Abhijit Sahay, Eunice E Santos, and Klaus Erik Schauser. 1993. Optimal broadcast and summation in the LogP model. In *Proceedings of the fifth annual ACM symposium on Parallel algorithms and architectures*. 142–153.
[22] David Kempe, Alin Dobra, and Johannes Gehrke. 2003. Gossip-based computation of aggregate information. In *44th Annual IEEE Symposium on Foundations of Computer Science, 2003. Proceedings.* IEEE, 482–491.
[23] Anne-Marie Kermarrec and Maarten Van Steen. 2007. Gossiping in distributed systems. *ACM SIGOPS operating systems review* 41, 5 (2007), 2–7.
[24] Jakub Konečnỳ, H Brendan McMahan, Felix X Yu, Peter Richtárik, Ananda Theertha Suresh, and Dave Bacon. 2016. Federated learning: Strategies for improving communication efficiency. *arXiv preprint arXiv:1610.05492* (2016).
[25] ChonLam Lao, Yanfang Le, Kshiteej Mahajan, Yixi Chen, Wenfei Wu, Aditya Akella, and Michael Swift. 2021. {ATP}: In-network aggregation for multi-tenant learning. In *18th USENIX Symposium on Networked Systems Design and Implementation (NSDI 21)*. 741–761.
[26] Tian Li, Anit Kumar Sahu, Ameet Talwalkar, and Virginia Smith. 2020. Federated learning: Challenges, methods, and future directions. *IEEE Signal Processing Magazine* 37, 3 (2020), 50–60.
[27] Samuel Madden, Michael J Franklin, Joseph M Hellerstein, and Wei Hong. 2002. TAG: A tiny aggregation service for ad-hoc sensor networks. *ACM SIGOPS Operating Systems Review* 36, SI (2002), 131–146.
[28] Samuel Madden, Michael J. Franklin, Joseph M. Hellerstein, and Wei Hong. 2003. TAG: a Tiny AGgregation service for ad-hoc sensor networks. *SIGOPS Oper. Syst. Rev.* 36, SI (dec 2003), 131–146. https://doi.org/10.1145/844128.844142
[29] Brendan McMahan, Eider Moore, Daniel Ramage, Seth Hampson, and Blaise Aguera y Arcas. 2017. Communication-efficient learning of deep networks from decentralized data. In *Artificial intelligence and statistics*. PMLR, 1273–1282.
[30] Shinji Motegi, Kiyohito Yoshihara, and Hiroki Horiuchi. 2006. DAG based in-network aggregation for sensor network monitoring. In *international Symposium on Applications and the Internet (SAINT'06)*. IEEE, 8–pp.
[31] Amirhossein Reisizadeh, Aryan Mokhtari, Hamed Hassani, Ali Jadbabaie, and Ramtin Pedarsani. 2020. Fedpaq: A communication-efficient federated learning method with periodic averaging and quantization. In *International conference on artificial intelligence and statistics*. PMLR, 2021–2031.

[32] Sujay Sanghavi, Bruce Hajek, and Laurent Massoulié. 2007. Gossiping with multiple messages. *IEEE Transactions on Information Theory* 53, 12 (2007), 4640–4654.
[33] Moshe Shaked and J George Shanthikumar. 2007. *Stochastic orders*. Springer.
[34] Christel Sirocchi and Alessandro Bogliolo. 2023. Community-based gossip algorithm for distributed averaging. In *IFIP International Conference on Distributed Applications and Interoperable Systems*. Springer, 37–53.
[35] David Wajc. 2017. Negative Association - Definition, Properties, and Applications. https://www.cs.cmu.edu/~dwajc/notes/Negative%20Association.pdf. [Online; accessed 08-July-2024].
[36] Bo Yin and Xuetao Wei. 2018. Communication-efficient data aggregation tree construction for complex queries in IoT applications. *IEEE Internet of Things Journal* 6, 2 (2018), 3352–3363.

A AGGREGATION DELAY ANALYSIS FOR ASYNCHRONOUS SETTING

In this section, we develop the analysis techniques used to prove Lemma 3 and 4 that were used to prove the near-optimality of OptSched in the Asynchronous setting.

First, we define a vector process $(\boldsymbol{X}^s)_{s=0}^{S}$ that tracks the progress that the different nodes have made in a schedule $\mathcal{S}$. Recall that each node locally has a notion of a sequence of virtual slots in its node schedule, even though there is no notion of synchronized slots across nodes in the Asynchronous setting. Let X_n^s denote the time at which node n has completed its virtual slot s. The sequence is initialized for virtual slot "0" as, $X_n^0 = 0$ for all nodes n.

The set of communications in slot s of $\mathcal{S}$ is $\mathcal{S}(s) = \left\{\left(n_{i,\text{tx}}^s, n_{i,\text{rx}}^s, m_i^s, k_i^s\right)\right\}_{i=1}^{I^s}$. Then, $2I^s$ nodes communicate in their virtual slot s, and the rest do "nothing". Let, $D_1^s, \ldots, D_{I^s}^s$ denote the respective link delays all distributed i.i.d., according to $F_{D(M)}$. Then, the process evolves as follows. For nodes n that do not communicate in slot s of $\mathcal{S}$, $X_n^s = X_n^{s-1}$. For nodes that communicate in slot s,

$$X_{n_{i,\text{rx}}^s}^s, X_{n_{i,\text{tx}}^s}^s = \max\left(X_{n_{i,\text{tx}}^s}^{s-1}, X_{n_{i,\text{rx}}^s}^{s-1}\right) + D_i^s, \quad \forall i \in [I^s], s \geq 1. \tag{6}$$

Since the aggregate needs to be computed at the server, the last communication in any schedule $\mathcal{S}$ will involve the server. Therefore, $T_{\text{async}}(\mathcal{S}) = X_1^S$.

The lower bound result of Lemma 3 is a straightforward application of Jensen's inequality.

Proof of Lemma 3. For nodes $n_{i,tx}^s$ and $n_{i,rx}^s$, we may use Jensen's inequality,

$$\mathbb{E}\left[X_{n_{i,\text{tx}}^s}^s\right], \mathbb{E}\left[X_{n_{i,\text{rx}}^s}^s\right] \geq \max\left(\mathbb{E}\left[X_{n_{i,tx}^s}^{s-1}\right], \mathbb{E}\left[X_{n_{i,\text{rx}}^s}^{s-1}\right]\right) + \mathbb{E}\left[D_i^s\right].$$

For nodes n that do not participate in slot s, $\mathbb{E}[X_n^s] = \mathbb{E}[X_n^{s-1}]$. By a nested application of Jensen's inequality, we can lower bound the aggregation delay of this system with link delay CDF $F_{D(M)}$ by another system with deterministic link delay $\mu_{D(M)}$. □

The technical challenge in obtaining an upper bound is that $X_{n_{i,\text{tx}}^s}^{s-1}$ and $X_{n_{i,\text{rx}}^s}^{s-1}$ in (6) maybe correlated due to past interactions between the nodes. The key insight in obtaining an upper bound is that the progress would be slower (i.e., the values of $\boldsymbol{X}^s$ would be higher) if they were independent. To that end, we define a bounding process as follows. Let $(\hat{D}^s)_{s=1}^{\infty}$ be an i.i.d., sequence of link delays each with CDF $F_{D(M)}$ Then, define a sequence of random variables $(\hat{X}^s)_{s=0}^{\infty}$ as,

$$\hat{X}^0 = 0, \qquad \hat{X}^s = \max\left(\hat{X}^{s-1}, \tilde{X}^{s-1}\right) + \hat{D}^s, \quad s \geq 1, \tag{7}$$

where $\tilde{X}^{s-1}$ is an i.i.d., copy of $\hat{X}^{s-1}$.

The following Lemma states that $\hat{X}^s$ is an "upper bound" on the time required for nodes to complete their virtual slot s, and is proved in Supplementary Material Section E.1.

Lemma 6. *For every $s \in [S]$, the vector $\left(X_n^s\right)_{n=1}^N$ is first-order-stochastically-dominated (FOSD) by the vector $\left(\hat{X}_n^s\right)_{n=1}^N$, where $\hat{X}_n^s$'s are i.i.d., copies of $\hat{X}^s$. That is, for any $x_1, \ldots, x_N \geq 0$,*
$P\left(X_1^s \leq x_1, \ldots, X_N^s \leq x_N\right) \geq \prod_{i=1}^N P\left(\hat{X}^s \leq x_i\right)$.

Due to Lemma 6, characterizing the convergence of $(\hat{X})_{s=0}^{\infty}$ gives an upper bound on the convergence of aggregation delay in the Asynchronous setting.

Lemma 7. *Let the process $\left(\hat{X}^s\right)_{s=0}^{\infty}$ be as defined in (7). Let the delay CDF $F_{D(M)}$ have a logMGF, with its associated right-tail Cramér function, $\Lambda_{D(M)}^*(\cdot)$. Let $\alpha^*(M) > 0$ be such that, $\Lambda_{D(M)}^*(\alpha^*(M)) = \ln 2$.*

Then, for any $\alpha > \alpha^*(M)$ *and any* $\mathrm{s} > 0$,

$$P\left(\hat{X}^{\mathrm{s}} \geq \alpha\,\mathrm{s}\right) \leq e^{-\mathrm{s}(\Lambda^*_{D(M)}(\alpha)-\ln 2)}. \tag{8}$$

Moreover, $\lim_{\mathrm{s}\to\infty} \frac{\hat{X}^{\mathrm{s}}}{\mathrm{s}} = \alpha^*(M)$ *in probability.*

The proof of Lemma 7 relies on the observation that $\hat{X}^{\mathrm{s}}$ can be interpreted approximately as the weighted height of a binary tree with s levels and i.i.d., edge lengths. With this observation, we use the result of [7] that characterizes the convergence of the heights of such trees to obtain our result. The Lemma is proved in Supplementary Material Section E.2. The tail bound on $T_{\mathrm{async}}(\mathcal{S})$ in Lemma 4 is a direct consequence of (8). The bound on $\mathbb{E}\left[T_{\mathrm{async}}(\mathcal{S})\right]$ may be obtained either from (8) or from the convergence in probability result and the observation that $\left(\hat{X}^s/s\right)_{s\geq 1}$ are Uniformly Integrable due to (8).

B NUMBER OF REPLICAS TO USE UNDER FAILURES

In this section, we explain our choice for the number of replicas K in (5) when running OptAgg. Consider the chunk-acceptance condition in Line 6 of OptAgg. Since $x/(1-x)$ is a convex function, by Jensen's inequality,

$$\sum_{k=1}^{K} \frac{A(m,k)}{1-A(m,k)} \geq K\frac{\bar{A}(m)}{1-\bar{A}(m)} \quad \text{a.s. where } \bar{A}(m) = \frac{1}{K}\sum_{k=1}^{K} A(m,k).$$

Due to the choice of $\sigma^2_{th} = \sigma^2$, a sufficient condition for the acceptance of chunk m is,

$$K \geq \frac{2(1-\bar{A}(m))}{\sigma^2\bar{A}(m)}. \tag{9}$$

By Lemma 10 in Supplementary Material G, the expectation of $\bar{A}(m)$ is, $\mathbb{E}[\bar{A}(m)] = (1-p)\,(1-p/2)^{\lceil \log N\rceil}$. Since, (9) is only a sufficient condition and the realizations of $\bar{A}(m)$ may vary around its mean, we did a search over choices of K of the following form with a tunable parameter c_K,

$$K = \frac{2(1-\mathbb{E}[\bar{A}(m)])}{c_K\sigma^2\,\mathbb{E}[\bar{A}(m)]}.$$

We found in simulations that $c_K = 2$ gave the lowest delay while satisfying the MSE constraint σ^2. This gives us the rule for selecting K as given in (5).

C OptSched PROTOCOL FOR GENERAL N

OptSched for a general number of nodes N is shown in Algorithm 3. The ideas here are the same as in the explanation in Section 3 for the special case of N being a power of 2. The difference is that identifying the nodes to aggregate the new chunk on in the Interleave and Merge step is more complicated as can be seen in Lines 12 to 19. The primary reason is that unlike in the simpler setting where the client nodes that participate in slot $l+1$ or later of $\mathcal{S}_{k'-1}$ was $N/2$, here the number of such nodes may be smaller than $N/2$. Therefore, the set of $\lfloor N/2\rfloor$ transmitters for the Interleave step need to be chosen more carefully so that the aggregation of chunk-replica $\phi^{-1}(k')$ in the Merge step does not collide with the aggregation of the first $k'-1$ chunk-replicas.

D PROOF OF LEMMA 2

Proof of Lemma 2. The proof will be by induction. Let $n_{k'}(\mathrm{s})$ represent the number of nodes that participate in slot s or later of schedule $\mathcal{S}_{k'}$.

Algorithm 3: OptSched $(N, \mathcal{M}, K)$

Input : Number of Nodes: N, (labelled 1 to N)
Chunk Indices: $\mathcal{M}$,
Replicas Per Chunk: K,

Output : Schedule $\mathcal{S}$ to compute $\sum_{n=2}^{N} \boldsymbol{u}^n(m,k)$ at node 1 for all $m \in \mathcal{M}, k \in [K]$.

Compute a bijective function, $\phi : \mathcal{M} \bigtimes [K] \rightarrow [|\mathcal{M}|\, K]$;
// Initialize Schedule
$N' = N$, s = 1 ;
while $N' > 1$ **do**
  $\mathcal{S}_1(\mathrm{s}) = \left\{(N' + 1 - j, j, \phi^{-1}(1)) : j = 1 \ldots \lfloor N'/2 \rfloor\right\}$;
  $N' = N' - |\mathcal{S}_1(\mathrm{s})|$;
  s = s +1 ;
end
for $k' = 2$ **to** $|\mathcal{M}|\, K$ **do**
  $\mathrm{S}_{k'-1} = \mathrm{len}(\mathcal{S}_{k'-1})$;
  $l = \mathrm{S}_{k'-1} - \lceil \log N \rceil + 1$;
  // Retain
  $\mathcal{S}_{k'}(\mathrm{s}) = \mathcal{S}_{k'-1}(\mathrm{s}), \quad 1 \le \mathrm{s} \le l$;
  // Interleave
  $\mathcal{N}_{\mathrm{tx}}$ = {client nodes that participate in slot $l + 2$ or later of $\mathcal{S}_{k'-1}$ } ;
  $\mathcal{N}^0_{\mathrm{rx}}$ = {nodes that do not participate in slot $l + 1$ or later of $\mathcal{S}_{k'-1}$ } ;
  $\mathcal{N}_{\mathrm{rem}} = [2, N] \setminus \mathcal{N}_{\mathrm{tx}} \setminus \mathcal{N}^0_{\mathrm{rx}}$;
  if $|\mathcal{N}_{\mathrm{tx}}| < \lfloor N/2 \rfloor$ **then**
    First add nodes from $\mathcal{N}_{\mathrm{rem}}$ to $\mathcal{N}_{\mathrm{tx}}$, and then, if needed, add nodes from $\mathcal{N}^0_{\mathrm{rx}}$ until $|\mathcal{N}_{\mathrm{tx}}| = \lfloor N/2 \rfloor$;
  end
  $\mathcal{N}_{\mathrm{rx}} = [1, N] \setminus \mathcal{N}_{\mathrm{tx}}$;
  $\mathcal{S}_{k'}(l+1) = \left\{\left(\mathcal{N}_{\mathrm{tx}}(i), \mathcal{N}_{\mathrm{rx}}(i), \phi^{-1}(k')\right) : j = 1 \ldots |\mathcal{N}_{\mathrm{tx}}|\right\}$;
  // Merge
  $\mathcal{N}^0 = [2, N] \setminus \mathcal{N}_{\mathrm{tx}}$; // nodes yet to transmit $\phi^{-1}(k')$
  for s = $l + 2$ **to** $\mathrm{S}_{k'-1}$ +1 **do**
    $\mathcal{N}_{\mathrm{avail}} = \mathcal{N}^0 \setminus$ {nodes participating in $\mathcal{S}_{k'-1}(\mathrm{s}-1)$} ;
    $\mathcal{S}^0 = \{\left(\mathcal{N}_{\mathrm{avail}}(|\mathcal{N}_{\mathrm{avail}}| + 1 - j), \mathcal{N}_{\mathrm{avail}}(j), \phi^{-1}(k')\right) : j = 1 \ldots \lfloor |\mathcal{N}_{\mathrm{avail}}|\, /2 \rfloor\}$;
    $\mathcal{S}_{k'}(\mathrm{s}) = \mathcal{S}_{k'-1}(\mathrm{s}-1) \cup \mathcal{S}^0$;
    $\mathcal{N}^0 = \mathcal{N}^0 \setminus$ { transmitters in $\mathcal{S}^0$} ;
  end
  // Final Aggregation
  $\mathcal{S}_{k'}(\mathrm{S}_{k'-1} + 2) = \{(\mathcal{N}^0(1), 1, \phi^{-1}(k'))\}$;
end

Induction Hypothesis (IH). For some $1 \leq k' \leq |\mathcal{M}|\,K$, the following statements are true for $\mathcal{S}_{k'}$.

(1) $\mathcal{S}_{k'}$ has induced an aggregation tree for each of the first k' chunk-replicas (ordered according to ϕ^{-1}).
(2) $\mathcal{S}_{k'}$ completes in $\mathrm{S}_{k'} = 2(k'-1) + \lceil \log N \rceil$ steps.
(3) $\mathcal{S}_{k'}$ does not violate communication constraints.
(4) $n_{k'}(l_{k'} + t) \leq 2^{\lceil \log N \rceil - t} \leq \lfloor N/2^{\mathrm{s}-1} \rfloor$, for all integers $1 \leq t \leq \lceil \log N \rceil - 1$, and where $l_{k'} = \mathrm{S}_{k'} - \lceil \log N \rceil + 1$.

Before proceeding further in the proof by induction, we remark how, if the induction is true, we obtain the statement of the Theorem. When the induction is run through till $k' = |\mathcal{M}|\,K$, Statement (3) ensures that the schedule $\mathcal{S}_{|\mathcal{M}|K}$ doesn't violate communication constraints, Statement (1) implies that $\mathcal{S}_{|\mathcal{M}|K}$ induces an aggregation tree for each of the $|\mathcal{M}|\,K$ chunk replicas, and Statement (2) ensures that $\mathcal{S}_{|\mathcal{M}|K}$ completes in the required time.

Base Case (BC). Here, we prove the Statements of IH for $\mathcal{S}_1$. Line 5 in Algorithm 3 ensures that once a node transmits, it no longer participates in $\mathcal{S}_1$. And, the While loop in Line 3 runs until the only node left to transmit is the server (Node 1). Therefore, $\mathcal{S}_1$ induces an aggregation tree for the first chunk-replica. This proves Statement (1).

To prove Statement (2), observe that if at the start of the while loop, $N' = 2$ nodes remain to transmit, then the schedule completes in one more slot (where, a communication between the two remaining nodes is scheduled). Now, assume that for some N' and all $N'' < N'$, if N'' nodes remain at the start of the While loop, then the schedule completes in $\lceil \log N'' \rceil$ more slots. Now, if N' nodes remain, then after one more iteration of the While loop, $\lceil N'/2 \rceil$ nodes will remain (see Lines 4 and 5 of Algorithm 3). So, when N' nodes remain, the number of slots needed to complete is $1 + \lceil \log \lceil N'/2 \rceil \rceil$, which is $\lceil \log N' \rceil$. This proves Statement (2) for the Base Case. Statement (3) is trivially true for the Base Case since Line 4 of Algorithm 3 ensures that a node participates in at most one communication in a slot.

To prove Statement (4) for the Base Case, notice that in the last slot, exactly one communication happens. Therefore, exactly two nodes participate in the last slot or later, leading to $n_1(\mathrm{S}_1) = 2$. Now, assume that for some $2 \leq \mathrm{s} \leq \lceil \log N \rceil - 1$, $n_1(l_1 + \mathrm{s}) \leq 2^{\lceil \log N \rceil - \mathrm{s}}$. Then, in the previous slot, at most $n_1(l_1 + \mathrm{s})$ more nodes may have been active, where each of them may have transmitted to one of the nodes still participating in slot $l_1 + \mathrm{s}$ or later. Therefore, $n_1(l_1 + \mathrm{s} - 1) \leq 2n_1(l_1 + \mathrm{s}) \leq 2^{\lceil \log N \rceil - \mathrm{s} + 1}$. This completes the proof of Statement (4).

Induction Step (IS):. Assuming the IH is true for $k' - 1$, here we prove the following statements for schedule $\mathcal{S}_{k'}$.

(1) $\mathcal{S}_{k'}$ has induced an aggregation tree for each of the first k' chunk replicas.
(2) $\mathcal{S}_{k'}$ completes in $\mathrm{S}_{k'} = 2(k'-1) + \lceil \log N \rceil$ steps.
(3) $\mathcal{S}_{k'}$ does not violate communication constraints.
(4) $n_{k'}(l_{k'} + \mathrm{s}) \leq 2^{\lceil \log N \rceil - \mathrm{s}} \leq \lfloor N/2^{\mathrm{s}-1} \rfloor$, for all integers $1 \leq \mathrm{s} \leq \lceil \log N \rceil - 1$, and where $l_{k'} = \mathrm{S}_{k'} - \lceil \log N \rceil + 1$.

To prove Statement 1, first, notice that $\mathcal{S}_{k'}$ contains all the communications that happen in $\mathcal{S}_{k'-1}$ (see Lines 11 and 24 of Algorithm 3). So, from Statement 1 of IH, $\mathcal{S}_{k'}$ also induces an aggregation tree for each of the first $k' - 1$ chunk replicas. It remains to prove that $\mathcal{S}_{k'}$ induces an aggregation tree for chunk replica k' too.

From Statement (4) of IH, no more than $\lfloor N/2 \rfloor$ nodes participate in slot $l_{k'-1} + 2$ or later of $S_{k'-1}$. Therefore, from Lines 12, 15 and 16 of Algorithm 3, $\mathcal{N}_{\mathrm{tx}}$ contains exactly $\lfloor N/2 \rfloor$ nodes. Then, Line 18 further implies that $\mathcal{N}_{\mathrm{rx}}$ contains exactly $\lceil N/2 \rceil$ nodes. Since $\mathcal{N}_{\mathrm{tx}}$ and $\mathcal{N}_{\mathrm{rx}}$ are disjoint, and

$|\mathcal{N}_{\text{tx}}| \leq |\mathcal{N}_{\text{rx}}|$, in the Interleave step (Line 19 of Algorithm 3), all nodes in $\mathcal{N}_{\text{tx}}$ transmit chunk replica k'.

Therefore, after the Interleave step, the number of client nodes yet to transmit k' is $|[2, N] \setminus \mathcal{N}_{\text{rx}}| = \lfloor \frac{N-1}{2} \rfloor$. Let n nodes be busy communicating the first $k'-1$ chunks in slot $l_{k'-1}+1$ of $\mathcal{S}_{k'-1}$. By definition, $n \leq n_{k'-1}(l_{k'-1}+1)$. Then, depending on whether $n \leq \lfloor N/2 \rfloor$ or not, from Lines 22 and 23 in the first loop of Algorithm 3, the number of nodes that transmit chunk replica k' in slot $l_{k'-1}+2$ of $\mathcal{S}_{k'}$ is,

$$\min\left\{\left\lfloor \frac{\lfloor \frac{N-1}{2} \rfloor}{2} \right\rfloor, \left\lfloor \frac{N-n}{2} \right\rfloor\right\}$$

So, the number of client nodes yet to transmit k' at the start of slot $l_{k'-1}+3$ of $\mathcal{S}_{k'}$ is,

$$\max\left\{\left\lceil \frac{\lfloor \frac{N-1}{2} \rfloor}{2} \right\rceil, \left\lfloor \frac{N-1}{2} \right\rfloor - \left\lfloor \frac{N-n}{2} \right\rfloor\right\}. \tag{10}$$

The above set of nodes forms a subset of $\mathcal{N}_{\text{rx}}$. And, from Statement (4) of IH, the number of nodes that participate in slot $l_{k'-1}+2$ or later of $\mathcal{S}_{k'-1}$ is at most $\lfloor N/2 \rfloor$, and they are all within the set $\mathcal{N}_{\text{tx}}$, which is disjoint from $\mathcal{N}_{\text{rx}}$. So, from slot $l_{k'-1}+3$ of $\mathcal{S}_{k'}$ onwards, all the nodes that are yet to transmit k' in any slot are available to communicate it. From the analysis of the Base Case, we know that if N' nodes want to aggregate a chunk replica at one node, then it takes $\lceil \log N' \rceil$ slots. So, upper bounding the number of nodes in (10),

$$\begin{aligned}
\max\left\{\left\lceil \frac{\lfloor \frac{N-1}{2} \rfloor}{2} \right\rceil, \left\lfloor \frac{N-1}{2} \right\rfloor - \left\lfloor \frac{N-n}{2} \right\rfloor\right\} &\leq \max\left\{\left\lceil \frac{N-1}{4} \right\rceil, \frac{N-1}{2} - \frac{N-n}{2} + \frac{1}{2}\right\}, \\
&= \max\left\{\left\lceil \frac{N-1}{4} \right\rceil, \frac{n}{2}\right\}, \\
&\leq \max\left\{\left\lceil \frac{N-1}{4} \right\rceil, 2^{\lceil \log N \rceil - 2}\right\}, \\
&\leq 2^{\lceil \log N \rceil - 2}.
\end{aligned}$$

Therefore, in the remaining $\lceil \log N \rceil - 2$ iterations of the loop in Line 21 of Algorithm 3, chunk-replica k' is aggregated at a single client node. Then, in the final aggregate step, this client node transmits to the server (see Line 27 of Algorithm 3). Therefore, $\mathcal{S}_{k'}$ constructs an aggregation tree for chunk replica k' too. This proves Statement (1) of the Induction Step.

Statement (2) is true for $\mathcal{S}_{k'}$ because it uses exactly 2 more slots than $\mathcal{S}_{k'-1}$, one in the Interleave Step, and one in the Final Aggregate Step (Lines 19 and 27 respectively of Algorithm 3). So, $\mathrm{S}_{k'} = \mathrm{S}_{k'-1} + 2 = 2(k'-1) + \lceil \log N \rceil$.

Now we prove Statement (3) for $\mathcal{S}_{k'}$. First, from IH, $\mathcal{S}_{k'-1}$ does not violate communication constraints. Since, $\mathcal{N}_{\text{tx}}$ and $\mathcal{N}_{\text{rx}}$ are disjoint sets, $\mathcal{S}_{k'}$ does not violate communication constraints in the Interleave Step (Line 19 of Algorithm 3). In the Merge steps, chunk replica k' is communicated only over those nodes that do not participate in $\mathcal{S}_{k'-1}$ in that slot (see Lines 22, 23, and 24). And, in the Final Aggregate step, only one communication happens. Therefore, $\mathcal{S}_{k'}$ does not violate any communication constraints.

The proof of Statement (4) is the same as the one for the Base Case. Notice that in the last slot, exactly one communication happens. Therefore, exactly two nodes participate in the last slot or later, leading to $n_{k'}(\mathrm{S}_{k'}) = 2$. Now, assume that for some $2 \leq \mathrm{s} \leq \lceil \log N \rceil - 1$, $n_{k'}(l_{k'} + \mathrm{s}) \leq 2^{\lceil \log N \rceil - \mathrm{s}}$. Then, in the previous slot, at most $n_{k'}(l_{k'} + \mathrm{s})$ more nodes may have been active, where each of

them may have transmitted to one of the nodes still participating in slot $l_{k'}$ + s or later. Therefore, $n_{k'}(l_{k'} + \mathrm{s} - 1) \leq 2n_{k'}(l_{k'} + \mathrm{s}) \leq 2^{\lceil \log N \rceil - \mathrm{s}+1}$. This completes the proof of Statement (4). □

E PROOFS OF SECTION A

E.1 Proof of Lemma 6

Proof of Lemma 6. Recall that $(n^{\mathrm{s}}_{1,\mathrm{tx}}, n^{\mathrm{s}}_{1,\mathrm{rx}}), \ldots, (n^{\mathrm{s}}_{I^{\mathrm{s}},\mathrm{tx}}, n^{\mathrm{s}}_{I^{\mathrm{s}},\mathrm{rx}})$ are the pairs of nodes that communicate in slot s. Let $n^{\mathrm{s}}_{1,0}, \ldots, n^{\mathrm{s}}_{N-2I^{\mathrm{s}},0}$ be the list of nodes not communicating in slot s.

Base Case: s = 1.

$$\begin{aligned}
P(X^1_1 \leq x_1, \ldots, X^1_N \leq x_N) &= P\left(\left(X^1_{n^1_{i,\mathrm{tx}}} \leq x^1_{n^1_{i,\mathrm{tx}}}, X^1_{n^1_{i,\mathrm{rx}}} \leq x^1_{n^1_{i,\mathrm{rx}}}\right)^{I^1}_{i=1}, \left(X^1_{n^1_{j,0}} \leq x^1_{n^1_{j,0}}\right)^{N-2I^1}_{j=1}\right), \\
&= \prod_{i=1}^{I^1} P\left(D \leq \min(x_{n^1_{i,\mathrm{tx}}}, x_{n^1_{i,\mathrm{rx}}})\right), \\
&\geq \prod_{n=1}^{N} P(D \leq x_n), \\
&= \prod_{n=1}^{N} P(\hat{X}^1 \leq x_n).
\end{aligned}$$

As the Induction Hypothesis, assume that the Theorem statement is true up to s − 1.

Induction Step: For notational convenience, define,

$$\begin{aligned}
Y^{\mathrm{s}}_{n^{\mathrm{s}}_{i,\mathrm{tx}}}, Y^{\mathrm{s}}_{n^{\mathrm{s}}_{i,\mathrm{rx}}} &= \max\left(X^{\mathrm{s}-1}_{n^{\mathrm{s}}_{i,\mathrm{tx}}}, X^{\mathrm{s}-1}_{n^{\mathrm{s}}_{i,\mathrm{rx}}}\right), \quad i = 1, \ldots, I^{\mathrm{s}}, \\
Y^{\mathrm{s}}_{n^{\mathrm{s}}_{j,0}} &= X^{\mathrm{s}-1}_{n^{\mathrm{s}}_{j,0}}, \quad j = 1, \ldots, N - 2I^{\mathrm{s}},
\end{aligned}$$

and,

$$\hat{Y}^{\mathrm{s}} = \max\left(\hat{X}^{\mathrm{s}-1}, \tilde{X}^{\mathrm{s}-1}\right). \tag{11}$$

Now,

$$\begin{aligned}
P(Y^{\mathrm{s}}_1 \leq y_1, \ldots, Y^{\mathrm{s}}_N \leq y_N) &= P\left(\left(Y^{\mathrm{s}}_{n^{\mathrm{s}-1}_{i,\mathrm{tx}}} \leq y_{n^{\mathrm{s}-1}_{i,\mathrm{tx}}}, Y^{\mathrm{s}}_{n^{\mathrm{s}-1}_{i,\mathrm{rx}}} \leq y_{n^{\mathrm{s}-1}_{i,\mathrm{rx}}}\right)^{I^{\mathrm{s}-1}}_{i=1}, \left(Y^{\mathrm{s}}_{n^{\mathrm{s}-1}_{j,0}} \leq y_{n^{\mathrm{s}-1}_{j,0}}\right)^{N-2I^{\mathrm{s}-1}}_{j=1}\right), \\
&= P\left(\left(Y^{\mathrm{s}}_{n^{\mathrm{s}-1}_{i,\mathrm{tx}}}, Y^{\mathrm{s}}_{n^{\mathrm{s}-1}_{i,\mathrm{rx}}} \leq \min\left(y_{n^{\mathrm{s}-1}_{i,\mathrm{tx}}}, y_{n^{\mathrm{s}-1}_{i,\mathrm{rx}}}\right)\right)^{I^{\mathrm{s}-1}}_{i=1}, \left(Y^{\mathrm{s}}_{n^{\mathrm{s}-1}_{j,0}} \leq y_{n^{\mathrm{s}-1}_{j,0}}\right)^{N-2I^{\mathrm{s}-1}}_{j=1}\right), \\
&= P\left(\left(X^{\mathrm{s}-1}_{n^{\mathrm{s}-1}_{i,\mathrm{tx}}}, X^{\mathrm{s}-1}_{n^{\mathrm{s}-1}_{i,\mathrm{rx}}} \leq \min\left(y_{n^{\mathrm{s}-1}_{i,\mathrm{tx}}}, y_{n^{\mathrm{s}-1}_{i,\mathrm{rx}}}\right)\right)^{I^{\mathrm{s}-1}}_{i=1}, \left(X^{\mathrm{s}-1}_{n^{\mathrm{s}-1}_{j,0}} \leq y_{n^{\mathrm{s}-1}_{j,0}}\right)^{N-2I^{\mathrm{s}-1}}_{j=1}\right), \\
&\overset{(a)}{\geq} \prod_{i=1}^{I^{\mathrm{s}-1}} P\left(\hat{X}^{\mathrm{s}-1} \leq \min\left(y_{n^{\mathrm{s}-1}_{i,\mathrm{tx}}}, y_{n^{\mathrm{s}-1}_{i,\mathrm{rx}}}\right)\right)^2 \prod_{j=1}^{N-2I^{\mathrm{s}-1}} P\left(\hat{X}^{\mathrm{s}-1} \leq y_{n^{\mathrm{s}-1}_{j,0}}\right), \\
&\overset{(b)}{=} \prod_{i=1}^{I^{\mathrm{s}-1}} P\left(\hat{Y}^{\mathrm{s}} \leq \min\left(y_{n^{\mathrm{s}}_{i,\mathrm{tx}}}, y_{n^{\mathrm{s}}_{i,\mathrm{rx}}}\right)\right) \prod_{j=1}^{N-2I^{\mathrm{s}-1}} \sqrt{P\left(\hat{Y}^{\mathrm{s}} \leq y_{i^{\mathrm{s}}_j}\right)}, \\
&\geq \prod_{n=1}^{N} P\left(\hat{Y}^{\mathrm{s}} \leq y_n\right).
\end{aligned} \tag{12}$$

(a) follows from the Induction Hypothesis, and (b) follows from the definition of $\hat{Y}$.

Further, by the same logic as in the base case,

$$\begin{aligned}
&P\left(\left(X_n^{\mathrm{s}} - Y_n^{\mathrm{s}} \le z_n\right)_{n=1}^{N}\right) \\
&\quad = P\left(\left(X^{\mathrm{s}}_{n^{\mathrm{s}}_{i,\mathrm{tx}}} - Y^{\mathrm{s}}_{n^{\mathrm{s}}_{i,\mathrm{tx}}} \le z_{n^{\mathrm{s}}_{i,\mathrm{tx}}}, X^{\mathrm{s}}_{n^{\mathrm{s}}_{i,\mathrm{rx}}} - Y^{\mathrm{s}}_{n^{\mathrm{s}}_{i,\mathrm{rx}}} \le z_{n^{\mathrm{s}}_{i,\mathrm{rx}}}\right)_{i=1}^{I^{\mathrm{s}}}, \left(X^{\mathrm{s}}_{n^{\mathrm{s}}_{j,0}} - Y^{\mathrm{s}}_{n^{\mathrm{s}}_{j,0}} \le z_{n^{\mathrm{s}}_{j,0}}\right)_{j=1}^{N-2I^{\mathrm{s}}}\right), \\
&\quad = \prod_{i=1}^{I^{\mathrm{s}}} P\left(D \le \min\left(z_{n^{\mathrm{s}}_{i,\mathrm{tx}}}, z_{n^{\mathrm{s}}_{i,\mathrm{rx}}}\right)\right), \\
&\quad \ge \prod_{n=1}^{N} P(D \le z_n), \\
&\quad = \prod_{n=1}^{N} P(\hat{X}^{\mathrm{s}} - \hat{Y}^{\mathrm{s}} \le z_n). \qquad (13)
\end{aligned}$$

From (12) and (13), the vectors $\left(Y_n^{\mathrm{s}}\right)_{n=1}^{N}$ and $\left(X_n^{\mathrm{s}} - Y_n^{\mathrm{s}}\right)_{n=1}^{N}$ are first-order-stochastically-dominated by vectors $\hat{Y}_n^{\mathrm{s}}$ and $(\hat{X}_n^{\mathrm{s}} - \hat{Y}_n^{\mathrm{s}})$ respectively (where $\hat{Y}_n^{\mathrm{s}}$'s are i.i.d., copies of $\hat{Y}^{\mathrm{s}}$). Further, $\left(X_n^{\mathrm{s}} - Y_n^{\mathrm{s}}\right)_{n=1}^{N}$ is independent of Y_n^{s} because for a node n, $X_n^{\mathrm{s}} - Y_n^{\mathrm{s}}$ is either 0 or is disributed according to $F_{D(M)}$ depending on whether it participates in slot s of the schedule or not, indpendent of the value of Y_n^{s}. And, $(\hat{X}_n^{\mathrm{s}} - \hat{Y}_n^{\mathrm{s}})$ is independent of $\hat{Y}_n^{\mathrm{s}}$ by construction.

Since FOSD is closed under addition of independent random vectors [33, Theorem 6.B.16], the vector $\left(X_n^{\mathrm{s}}\right)_{n=1}^{N}$ is first-order-statistically-dominated by $\left(\hat{X}_n^{\mathrm{s}}\right)_{n=1}^{N}$. That is,

$$P\left(X_1^{\mathrm{s}} \le x_1, \ldots, X_N^{\mathrm{s}} \le x_N\right) \ge \prod_{i=1}^{N} P\left(\hat{X}^{\mathrm{s}} \le x_i\right).$$

□

E.2 Proof of Lemma 7

Proof of Lemma 7. This proof is inspired from [7]. Recall $\hat{Y}^s$ from (11).

Consider an infinite binary tree B_∞ where each edge length is distributed according to the CDF $F_{D(M)}$. Observe that, in distribution, $\hat{Y}^{\mathrm{s}+1}$ is the length of the longest path from the root to a node at depth s in B_∞. Denote, $\left\{Z_i^{\mathrm{s}}\right\}_{i=1}^{2^{\mathrm{s}}}$ as the lengths of the 2^{s} paths from the root to the nodes at depth s. Z_i is a sum of s i.i.d., random variables distributed according to $F_{D(M)}$, but Z_i may depend on Z_j, for $i \ne j$.

We first prove (8). By using Chernoff bound, we get,

$$P(Z_i^{\mathrm{s}} \ge \alpha\,\mathrm{s}) \le e^{-\mathrm{s}\,\Lambda_D^*(\alpha)} \quad , \forall i \in [2^{\mathrm{s}}].$$

By a union bound over all the Z_i^{s}'s,

$$\begin{aligned}
P(\hat{Y}^{\mathrm{s}+1} \ge \alpha\,\mathrm{s}) &= P\left(\max_{i=1}^{2^{\mathrm{s}}} Z_i^{\mathrm{s}} \ge \alpha\,\mathrm{s}\right), \\
&\le 2^{\mathrm{s}} P(Z_1^{\mathrm{s}} \ge \alpha\,\mathrm{s}), \\
&\le 2^{\mathrm{s}} e^{-\mathrm{s}\,\Lambda_D^*(\alpha)}, \\
&= e^{-\mathrm{s}(\Lambda_D^*(\alpha) - \ln 2)}.
\end{aligned}$$

Since, $\hat{Y}^{\mathrm{s}+1} = \max\{\hat{X}^{\mathrm{s}}, \tilde{X}^{\mathrm{s}}\}$, $P(\hat{X}^{\mathrm{s}} \ge \alpha\,\mathrm{s}) \le P(\hat{Y}^{\mathrm{s}+1} \ge \alpha\,\mathrm{s})$. Therefore, we get,

$$P(\hat{X}^{\mathrm{s}} \ge \alpha\,\mathrm{s}) \le e^{-\mathrm{s}(\Lambda_D^*(\alpha) - \ln 2)}. \qquad (14)$$

Next, we prove the convergence in probability. Since, $\Lambda_D^*(\alpha) > \ln 2$ for all $\alpha > \alpha^*$, from (14) we get,

$$\lim_{s\to\infty} P\left(\frac{\hat{X}^s}{s} \geq \alpha\right) = 0. \tag{15}$$

In order to prove the upper bound, we make use of Galton-Watson (GW) Processes [14].

Definition 2. *A Galton-Watson (GW) process $(Z_r)_{r\in\mathbb{Z}_+}$ with offspring distribution $(p_k)_{k\in\mathbb{Z}_+}$ is a discrete-time Markov chain taking values in the set $\mathbb{Z}_+$ of non-negative integers whose transition probabilities are as follows,*

$$P\left(Z_{r+1} = k | Z_r = m\right) = p_k^{*m},$$

where $\left(p_k^{*m}\right)_{k\in\mathbb{Z}_+}$ is the m^{th} power distribution of $(p_k)_{k\in\mathbb{Z}_+}$. In other words, given $Z_r = m$, Z_{r+1} is the sum of m i.i.d., random variables each with distribution $(p_k)_{k\in\mathbb{Z}_+}$.

The GW process maybe visualized as a tree, where there is a root node, and the number of children that each node in this tree has is distributed independently according to $(p_k)_{k\in\mathbb{Z}_+}$. Then, Z_r may be interpreted as the number of nodes in this tree at level r.

We will use the following fact [14, Theorem 6.1] in the proof.

Fact 1. *If the mean of the offspring distribution of a GW process $(Z_r)_{r\in\mathbb{Z}_+}$ is greater than 1, i.e., $\sum_k kp_k > 1$, then, there exists, $0 < q \leq 1$, such that, the process survives with probability q. That is,*

$$P\left(\bigcap_{r\geq 0} Z_r > 0\right) = q.$$

In other words, the tree produced by the GW process is infinite with probability q.

Consider the infinite binary tree B_∞ where the edge lengths are i.i.d., distributed according to $F_{D(M)}$. Recall that $\hat{Y}^{s+1}$ is the maximum length of a path from the root to a node at depth s. Let l, t be positive integers, and let $0 < \alpha' < \alpha^*$. In order to differentiate between nodes in B_∞ and nodes in the GW process, we will call the nodes in the former as, simply, nodes, and in the latter as gw-nodes.

We will start a GW process from each node at depth t. The node at depth t is a gw-node. Then, for each node that is also a gw-node, another node at depth l from it in B_∞ is its child in the GW process if the corresponding path length is at least $\alpha' l$. Letting $D_1, \ldots, D_l$ denote i.i.d., random variables with CDF $F_{D(M)}$, the expected number of children of a gw-node may then be calculated as,

$$\begin{aligned} 2^l P\left(\sum_{j=1}^{l} D_j \geq \alpha' l\right) &= 2^l e^{-l\Lambda^*(\alpha')+o(l)}, \\ &> 1, \quad \text{for large enough } l. \end{aligned}$$

The first statement follows from large deviations, and since there are 2^l nodes at depth l in a binary tree which are candidates to be children in the GW process. The second statement follows since for any $\alpha' < \alpha^*$, we have $\Lambda^*(\alpha') < \ln 2$.

Since the expected number of children in the GW process is greater than 1, from Fact 1 we have that the process survives with a positive probability q. Since we started a GW process at each node at depth t in B_∞, there are 2^t independent GW processes that each survive with probability q. Therefore, the probability that any one of them survives is $1 - (1-q)^{2^t}$.

Since $\hat{X}^s \geq \hat{Y}^s$, picking a path along one of these surviving processes, we get,

$$P\left(\frac{\hat{X}^s}{s} \geq \alpha' \frac{s-1-t}{s}\right) = 1 - (1-q)^{2^t} \quad , \forall\, s \geq t.$$

Taking the limit,

$$\lim_{s\to\infty} P\left(\frac{\hat{X}^s}{s} \geq \alpha'\right) = \lim_{s\to\infty} P\left(\frac{\hat{X}^s}{s} \geq \alpha' \frac{s-1-t}{s}\right),$$
$$= 1-(1-q)^{2^t}.$$

The above probability can be made arbitrarily close to 1 by making t large. That is,

$$\lim_{s\to\infty} P\left(\frac{\hat{X}^s}{s} \geq \alpha'\right) = 1. \tag{16}$$

From (15) and (16) we conclude,

$$\lim_{s\to\infty} \frac{\hat{X}^s}{s} = \alpha^* \quad \text{in probability.}$$

□

F PROOF OF LEMMA 5

We split the statement of Lemma 5 into two parts, the first part on the unbiased property of the estimate $\hat{u}(m)$, the second part on the bound on the MSE.

Part 1:

The unbiased property follows from a straightforward application of linearity of expectation.

Lemma 8. *Under the same system condition as in the statement of Theorem 5, $\hat{\boldsymbol{u}}_\beta(m)$ is an unbiased estimate of $\boldsymbol{u}(m)$, i.e.,*

$$\mathbb{E}\left[\hat{\boldsymbol{u}}_\beta(m)\middle|\boldsymbol{u}(m)\right] = \boldsymbol{u}(m).$$

Proof. Consider chunk m. Recall the definition of $\mathcal{F}_{\text{drop}}(m)$ from (4).

$$\begin{aligned}
\mathbb{E}\left[\hat{\boldsymbol{u}}_{\beta}(m)\middle|\, \boldsymbol{u}(m), \mathcal{F}_{\text{drop}}(m)\right] &= \mathbb{E}\left[\sum_{k=1}^{K} \frac{\beta(m,k)}{(N-1)A(m,k)} \sum_{n\in\mathcal{A}(m,k)} \boldsymbol{u}^n(m)\middle| \boldsymbol{u}(m), \mathcal{F}_{\text{drop}}(m)\right], \\
&= \mathbb{E}\left[\sum_{k=1}^{K} \frac{\beta(m,k)}{(N-1)A(m,k)} \sum_{n=2}^{N} \boldsymbol{u}^n(m)\, \mathbb{1}_{\{n\in\mathcal{A}(m,k))\}} \middle| \boldsymbol{u}(m), \mathcal{F}_{\text{drop}}(m)\right], \\
&= \mathbb{E}\left[\sum_{n=2}^{N} \boldsymbol{u}^n(m) \sum_{k=1}^{K} \frac{\beta(m,k)}{(N-1)A(m,k)}\, \mathbb{1}_{\{n\in\mathcal{A}(m,k)\}} \middle| \boldsymbol{u}(m), \mathcal{F}_{\text{drop}}\right], \\
&= \mathbb{E}\Bigg[\mathbb{E}\Bigg[\sum_{n=2}^{N} \boldsymbol{u}^n(m) \sum_{k=1}^{K} \frac{\beta(m,k)}{(N-1)A(m,k)}\, \mathbb{1}_{\{n\in\mathcal{A}(m,k)\}} \\
&\qquad\qquad \Bigg| (\boldsymbol{u}^n(m))_{n=2}^{N}, \mathcal{F}_{\text{drop}}(m)\Bigg]\Bigg| \boldsymbol{u}(m), \mathcal{F}_{\text{drop}}(m)\Bigg], \\
&\overset{(a)}{=} \mathbb{E}\left[\sum_{n=2}^{N} \boldsymbol{u}^n(m) \sum_{k=1}^{K} \frac{\beta(m,k)}{(N-1)A(m,k)} A(m,k)\, \mathbb{1}_{\{A(m,k)>0\}} \middle| \boldsymbol{u}(m), \mathcal{F}_{\text{drop}}(m)\right], \\
&\overset{(b)}{=} \mathbb{E}\left[\boldsymbol{u}(m) \sum_{k=1}^{K} \beta(m,k)\, \mathbb{1}_{\{A(m,k)>0\}} \middle| \boldsymbol{u}(m), \mathcal{F}_{\text{drop}}(m)\right], \\
&= \boldsymbol{u}(m) \sum_{k=1}^{K} \frac{\mathbb{1}_{\{A(m,k)>0\}}\, \frac{A(m,k)}{1-A(m,k)}}{\sum_{j=1}^{K} \frac{A(m,j)}{1-A(m,j)}}, \\
&= \boldsymbol{u}(m),
\end{aligned}$$

where (a) follows from symmetry in the link failure model, and (b) follows from the definition of $\beta(m,k)$'s.

By the Tower rule of expectations,

$$\begin{aligned}
\mathbb{E}\left[\hat{\boldsymbol{u}}_{\beta}(m)\middle|\, \boldsymbol{u}(m)\right] &= \mathbb{E}\left[\mathbb{E}\left[\hat{\boldsymbol{u}}(m)\middle| \boldsymbol{u}(m), \mathcal{F}_{\text{drop}}\right] \middle| \boldsymbol{u}(m)\right], \\
&= \boldsymbol{u}(m).
\end{aligned}$$

Since $\beta^*(m,k)$'s satisfy all the constraints of $\beta(m,k)$'s, $\hat{\boldsymbol{u}}(m)$ is an unbiased estimate as well. □

Part 2:

Let $\mathcal{F}_{\text{drop}}(m)$ be as defined in (4). For succinctness, here we only show the proof of the MSE bound for $\beta^*(m,k)$'s. However, one can observe that stopping the equation (18) at (b) would prove the MSE bound for $\beta(m,k)$'s.

Lemma 9. *Under the same system condition as in the statement of Lemma 5, we have,*

$$\mathbb{E}\left[\|\hat{\boldsymbol{u}}(m) - \boldsymbol{u}(m)\|_2^2 \middle| (\boldsymbol{u}^n(m))_{n=2}^{N}, \mathcal{F}_{\text{drop}}(m)\right] \le \frac{1}{(N-1)^2} \frac{2}{\sum_{k=1}^{K} \frac{A(m,k)}{1-A(m,k)}} \sum_{n=2}^{N} \|\boldsymbol{u}^n(m)\|_2^2 .$$

The proof of Lemma 9 uses the property of *negative association* (NA) of random variables which we introduce briefly.

Definition 3 ([19]). *A set of random variables $X^1, X^2, \ldots, X^l$ are said to be negatively associated (NA) if, for every pair of disjoint subsets S_1, S_2 of $\{1, \ldots, l\}$, $\mathrm{cov}(f(X^i, I \in S_1), g(X^j, j \in S_2)) \leq 0$ for all non-decreasing functions f, g.*

Some examples of sets of random variables that have the NA property are stated below. See [35] for a proof.

Proposition 1. *(1) If random variables $X^1, X^2, \ldots, X^l$ are random permutations of $x^1, x^2, \ldots, x^l$, with each permutation being equally likely, then they are NA.*
(2) If $X^1, X^2, \ldots, X^l$ are NA, and $Y^1, Y^2, \ldots, Y^r$ are NA, and further $\{X_i\}_i$ and $\{Y_j\}_j$ are independent of each other, then $X^1, \ldots, X^l, Y^1, \ldots, Y^r$ are NA.
(3) Let $f_1, \ldots, f_r : \mathbb{R}^l \to \mathbb{R}$ be either all monotonically increasing or all monotonically decreasing in each of their input coordinates, with each f_i operating on disjoint subsets of $[l]$, $S_1, S_2, \ldots, S_r \subset [l]$. Let, $\boldsymbol{X} \triangleq \left(X^i\right)_{i=1}^l$. Then, $Y^1 = f_1(\boldsymbol{X}), \ldots, Y^r = f_r(\boldsymbol{X})$ are NA if $X^1, \ldots, X^l$ are NA.

The property of negative association imbues several other useful properties on the set of random vectors. We shall only use the following property here. See [35] for a proof.

Proposition 2. *Let the set of random variables $X^1, X^2, \ldots, X^l$ be negatively associated. Then,*

$$\mathbb{E}\left[\left(\sum_{i=1}^{l} X^i - \mathbb{E}\left[\sum_{i=1}^{l} X^i\right]\right)^2\right] \leq \sum_{i=1}^{l} \mathbb{E}\left[\left(X^i - \mathbb{E}[X^i]\right)^2\right].$$

We are prepared to prove Lemma 9.

Proof of Lemma 9. First, define a "fictional quantity" for every Client n,

$$\hat{\boldsymbol{u}}^n(m) \triangleq \boldsymbol{u}^n(m) \sum_{k=1}^{K} \beta^*(m,k) \frac{\mathbb{1}_{\{n \in \mathcal{A}(m,k)\}}}{A(m,k)}.$$

Observe that,

$$\hat{\boldsymbol{u}}(m) = \frac{1}{N-1} \sum_{n=2}^{N} \hat{\boldsymbol{u}}^n(m). \tag{17}$$

From the above equation, $\hat{\boldsymbol{u}}^n(m)$ maybe interpreted as an estimate of $\boldsymbol{u}^n(m)$, although the server only has knowledge of $\hat{\boldsymbol{u}}(m)$ and not of $\hat{\boldsymbol{u}}^n(m)$. The MSE of $\hat{\boldsymbol{u}}^n(m)$ is bounded as,

$$\begin{aligned}
&\mathbb{E}\left[\|\hat{\boldsymbol{u}}^n(m) - \boldsymbol{u}^n(m)\|_2^2 \,|\, \boldsymbol{u}^n(m), \mathcal{F}_{\text{drop}}(m)\right] \\
&\quad = \|\boldsymbol{u}^n(m)\|_2^2 \, \mathbb{E}\left[\left(\sum_{k=1}^{K} \frac{\beta^*(m,k)}{A(m,k)} \left(\mathbb{1}_{\{n \in \mathcal{A}(m,k)\}} - A(m,k)\right)\right)^2 \middle| \mathcal{F}_{\text{drop}}(m)\right], \\
&\quad \overset{(a)}{=} \|\boldsymbol{u}^n(m)\|_2^2 \sum_{k=1}^{K} \left(\frac{\beta^*(m,k)}{A(m,k)}\right)^2 \mathbb{E}\left[\left(\mathbb{1}_{\{n \in \mathcal{A}(m,k)\}} - A(m,k)\right)^2 \middle| \mathcal{F}_{\text{drop}}(m)\right], \\
&\quad \overset{(b)}{=} \|\boldsymbol{u}^n(m)\|_2^2 \sum_{k=1}^{K} \frac{\beta^*(m,k)^2}{A(m,k)^2} A(m,k)(1 - A(m,k)), \\
&\quad \overset{(c)}{=} \|\boldsymbol{u}^n(m)\|_2^2 \, \frac{1}{\sum_{k=1}^{K} \frac{A(m,k)}{1 - A(m,k)}}.
\end{aligned} \tag{18}$$

(a) follows because, under the link failure model, links fail independently across trees. Therefore, node drops are also independent across trees. (c) follows from the choice of $\beta^*(m,k)$ as in Algorithm 2. In fact, it is the value of β_k's that minimizes the quantity in (b).

To extend (18) to our intended result, we need to establish a set of NA random variables.

By symmetry, given $\mathcal{F}_{\text{drop}}(m)$, $\mathbb{1}_{\{1\in\mathcal{A}(m,k)\}},\dots,\mathbb{1}_{\{N\in\mathcal{A}(m,k)\}}$ are uniformly random permutations of $|\mathcal{A}(m,k)|$ number of 1's, and $N-1-|\mathcal{A}(m,k)|$ number 0's. Therefore, by Statement 1 of Proposition 1, they are NA.

Since link failures are independent across trees, for any $k_1 \neq k_2$, the set of random variables $\left\{\mathbb{1}_{\{n\in\mathcal{A}_{k_1}\}}\right\}_{n=2}^{N}$ is independent of the set $\left\{\mathbb{1}_{\{n\in\mathcal{A}_{k_2}\}}\right\}_{n=2}^{N}$. Therefore by Statement 2 of Proposition 1, given $\mathcal{F}_{\text{drop}}(m)$, $\bigcup_{k=1}^{K}\left\{\mathbb{1}_{\{n\in\mathcal{A}(m,k)\}}\right\}_{n=2}^{N}$ is a set of NA random variables.

Consider a coordinate $c\in[d]$. Let, $S_c^P \triangleq \{n\in[2,N] : u_c^n(m)\geq 0\}$ be the set of nodes where the c^{th} coordinate of their update is non-negative. Recalling that,

$$\hat{u}_c^n(m) = u_c^n(m)\sum_{k=1}^{K}\beta^*(m,k)\frac{\mathbb{1}_{\{n\in\mathcal{A}(m,k)\}}}{A(m,k)},$$

given $\mathcal{F}_{\text{drop}}(m)$ and $\boldsymbol{u}^n(m)$, $\hat{u}_c^n(m)$ is a monotonically increasing function of $\left(\mathbb{1}_{\{n\in\mathcal{A}(m,k)\}}\right)_{k=1}^{K}$ for every $n\in S_c^P$. Moreover, for distinct $n_1,n_2\in S_c^P$, $\hat{u}_c^{n_1}$ and $\hat{u}_c^{n_2}$ operate on disjoint subsets of $\bigcup_{k=1}^{K}\left\{\mathbb{1}_{\{n\in\mathcal{A}(m,k)\}}\right\}_{n=2}^{N}$. Therefore, by Statement 3 of Proposition 1, $\left\{\hat{u}_c^n\right\}_{n\in S_c^P}$ is a set of NA random variables given $\mathcal{F}_{\text{drop}}(m)$ and $(\boldsymbol{u}^n(m))_{n=2}^{N}$. And, from Proposition 2, we have,

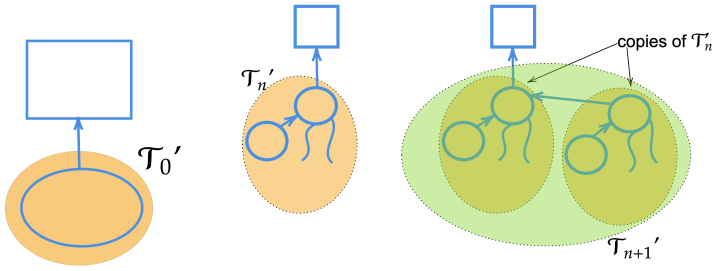
$$\begin{aligned}\mathbb{E}\left[\left(\sum_{n\in S_c^P}\hat{u}_c^n(m)-\sum_{n\in S_c^P}u_c^n(m)\right)^2\middle|\mathcal{F}_{\text{drop}}(m),(\boldsymbol{u}^n(m))_{n=2}^{N}\right]\\ \leq\sum_{n\in\mathcal{S}_c^P}\mathbb{E}\left[(\hat{u}_c^n(m)-u_c^n(m))^2\middle|\mathcal{F}_{\text{drop}}(m),(\boldsymbol{u}^n(m))_{n=2}^{N}\right].\end{aligned}\tag{19}$$

Let $S_c^N \triangleq \left\{n\in[,2N] : u_c^n<0\right\}$ be the set of nodes where the c^{th} coordinate is negative. By the same reasoning as above (except that $\hat{u}_c^n$ is a decreasing function of $\left(\mathbb{1}_{\{n\in\mathcal{A}(m,k)\}}\right)_{k=1}^{K}$), we have,

$$\begin{aligned}\mathbb{E}\left[\left(\sum_{n\in S_c^N}\hat{u}_c^n(m)-\sum_{n\in S_c^N}u_c^n(m)\right)^2\middle|\mathcal{F}_{\text{drop}}(m),(\boldsymbol{u}^n(m))_{n=2}^{N}\right]\\ \leq\sum_{n\in\mathcal{S}_c^N}\mathbb{E}\left[(\hat{u}_c^n(m)-u_c^n(m))^2\middle|\mathcal{F}_{\text{drop}}(m),(\boldsymbol{u}^n(m))_{n=2}^{N}\right].\end{aligned}\tag{20}$$

Recalling the identity, $(x+y)^2\leq 2x^2+2y^2$, we have,

$$\begin{aligned}&\mathbb{E}\left[\left(\sum_{n=2}^{N}\hat{u}_c^n(m)-\sum_{n=2}^{N}u_c^n(m)\right)^2\middle|\mathcal{F}_{\text{drop}}(m),(\boldsymbol{u}^n(m))_{n=2}^{N}\right]\\ &\quad\leq 2\,\mathbb{E}\left[\left(\sum_{n\in S_c^P}\hat{u}_c^n(m)-\sum_{n\in S_c^P}u_c^n(m)\right)^2\middle|\mathcal{F}_{\text{drop}}(m),(\boldsymbol{u}^n(m))_{n=2}^{N}\right]\\ &\qquad+2\,\mathbb{E}\left[\left(\sum_{n\in S_c^N}\hat{u}_c^n(m)-\sum_{n\in S_c^N}u_c^n(m)\right)^2\middle|\mathcal{F}_{\text{drop}}(m),(\boldsymbol{u}^n(m))_{n=2}^{N}\right],\\ &\quad\overset{(a)}{\leq} 2\sum_{n=2}^{N}\mathbb{E}\left[(\hat{u}_c^n(m)-u_c^n(m))^2\middle|\mathcal{F}_{\text{drop}}(m),(\boldsymbol{u}^n(m))_{n=2}^{N}\right],\end{aligned}\tag{21}$$



(a) (b)

Fig. 9

where (a) follows from (19) and (20). Now, we are prepared to prove the statement of the Lemma,

$$
\begin{aligned}
&\mathbb{E}\left[\|\hat{\boldsymbol{u}}(m)-\boldsymbol{u}(m)\|_2^2 \,\middle|\, (\boldsymbol{u}^n(m))_{n=2}^N, \mathcal{F}_{\text{drop}}(m)\right] \\
&\quad\overset{(a)}{=} \mathbb{E}\left[\left\|\frac{1}{N-1}\left(\sum_{n=2}^N \hat{\boldsymbol{u}}^n(m)-\sum_{n=2}^N \boldsymbol{u}^n(m)\right)\right\|_2^2 \,\middle|\, (\boldsymbol{u}^n(m))_{n=2}^N, \mathcal{F}_{\text{drop}}(m)\right], \\
&\quad= \frac{1}{(N-1)^2}\sum_{c=1}^d \mathbb{E}\left[\left(\sum_{n=2}^N \hat{u}_c^n(m)-\sum_{n=2}^N u_c^n(m)\right)_2^2 \,\middle|\, (\boldsymbol{u}^n(m))_{n=2}^N, \mathcal{F}_{\text{drop}}(m)\right], \\
&\quad\overset{(b)}{\leq} \frac{1}{(N-1)^2}\sum_{c=1}^d 2\sum_{n=2}^N \mathbb{E}\left[(\hat{u}_c^n(m)-u_c^n(m))^2 \,\middle|\, \mathcal{F}_{\text{drop}}(m), (\boldsymbol{u}^n(m))_{n=2}^N\right], \\
&\quad= \frac{2}{(N-1)^2}\sum_{n=2}^N \mathbb{E}\left[\|\hat{\boldsymbol{u}}^n(m)-\boldsymbol{u}^n(m)\|_2^2 \,\middle|\, (\boldsymbol{u}^n(m))_{n=2}^N, \mathcal{F}_{\text{drop}}(m)\right], \\
&\quad\overset{(c)}{=} \frac{2}{(N-1)^2}\frac{1}{\sum_{k=1}^K \frac{A(m,k)}{1-A(m,k)}}\sum_{n=2}^N \|\boldsymbol{u}^n(m)\|^2 .
\end{aligned}
$$

(a) is true due to (17). (b) follows from (21). (c) follows from (18). □

G EXPECTATION AND VARIANCE OF $A(m,k)$

G.1 Definitions

Definition 4. *Define a sequence of* Standard Trees $(\mathcal{T}_n)_{n\geq 1}$, *where* $\mathcal{T}_n$ *consists of* 2^n+1 *nodes (including clients and server), while denoting* $\mathcal{T}_n'$ *as the subtree of* $\mathcal{T}_n$ *that contains all the clients, but not the server.* $\mathcal{T}_0$ *is defined as shown in Fig. 9a, and the further trees in the sequence are derived recursively as shown in Fig. 9b.*

G.2 Tracking Node Drop Rate

Lemma 10. *Consider that a Standard Tree $\mathcal{T}_n$ with $2^n + 1$ nodes (including the server), has nodes failing at rate p. Then, denoting $F^{n,p}$ as the fraction of nodes that drop,*

$$\mathbb{E}\left[F^{n,p}\right] = 1 - (1-p)\left(1 - \frac{p}{2}\right)^n,$$
$$\mathrm{var}(F^{n,p}) = \frac{p(1-p)(3-p)}{4}\left(1-\frac{p}{2}\right)^{2n-1} - \frac{p(1-p)}{4}\left(\frac{1-\frac{p}{2}}{2}\right)^{n+1}.$$

Proof. Define X_n as the random variable denoting the number of node drops in the standard tree $\mathcal{T}_n$. Denoting the client node that directly communicates with the server in a Standard Tree as the *root client*, define Y_n as the number of node drops in $\mathcal{T}_n$ due to failures of all client nodes except the root client.

Observe that since $\mathcal{T}_1$ only has 2 clients, one of which is the root, Y_1 is a Bern(p) random variable. Then, let Z_n be a independent Bern(p) random variable and Y_n' be an independent copy of Y_n. Due to the recursive construction of $\mathcal{T}_n$, we have,

$$Y_{n+1} \overset{d}{=} Y_n + Z_n 2^n + (1-Z_n)Y_n', \quad n \geq 1, \tag{22}$$

where $\overset{d}{=}$ means equal in distribution. Defining, $y_n = \mathbb{E}[Y_n]/2^n$ and simplifying,

$$y_{n+1} = y_n\left(1 - \frac{p}{2}\right) + \frac{p}{2}.$$

Solving the equation with the initial condition, $y_1 = \mathbb{E}[Y_1]/2 = p/2$ (because Y_1 is a Bern(p) random variable), we get,

$$y_n = 1 - \left(1 - \frac{p}{2}\right)^n. \tag{23}$$

Further, observe that,

$$X_n \overset{d}{=} Z_n 2^n + (1-Z_n)Y_n. \tag{24}$$

Then, since $F^{n,p} = X_n/2^n$, we get,

$$\mathbb{E}[F^{n,p}] = 1 - (1-p)\left(1 - \frac{p}{2}\right)^n. \tag{25}$$

From (22),

$$\mathbb{E}[Y_{n+1}^2] = \mathbb{E}[Y_n^2] + 2^{2n}\,\mathbb{E}[Z_n^2] + \mathbb{E}[Y_n^2]\,\mathbb{E}[(1-Z_n)^2] + 2^{n+1}\,\mathbb{E}[Y_n]\,\mathbb{E}[Z_n] + 2\,\mathbb{E}[(1-Z_n)]\,\mathbb{E}[Y_n]^2.$$

Define, $y_n^{(2)} \triangleq \mathbb{E}[Y_n^2]/2^{2n}$, and recalling that $y_n = \mathbb{E}[Y_n]/2^n$, we get,

$$y_{n+1}^{(2)} = \frac{\left(1-\frac{p}{2}\right)}{2} y_n^{(2)} + \frac{p}{2} y_n + \frac{(1-p)}{2} y_n^2 + \frac{p}{4}.$$

Plugging in the value of y_n from (23),

$$y_{n+1}^{(2)} = \frac{\left(1-\frac{p}{2}\right)}{2} y_n^{(2)} + \frac{(1-p)}{2}\left(1-\frac{p}{2}\right)^{2n} - \left(1-\frac{p}{2}\right)^{n+1} + \frac{1}{2} + \frac{p}{4}.$$

Solving for $y_n^{(2)}$ using the above iteration and the initial condition $y_1^{(2)} = p/4$,

$$y_n^{(2)} = 1 - 2\left(1-\frac{p}{2}\right)^n + \left(1-\frac{p}{2}\right)^{2n-1} - \frac{p}{4}\left(\frac{1-\frac{p}{2}}{2}\right)^{n-1}.$$

Observing that, $\mathbb{E}[(F^{n,p})^2] = \mathbb{E}[X_n^2]/2^{2n}$. Then, from (24),

$$\begin{aligned}\mathbb{E}[(F^{n,p})^2] &= p + (1-p)y_n^{(2)},\\ &= 1 - 2(1-p)\left(1-\frac{p}{2}\right)^n + (1-p)\left(1-\frac{p}{2}\right)^{2n-1} - \frac{p(1-p)}{4}\left(\frac{1-\frac{p}{2}}{2}\right)^{n-1}.\end{aligned}$$

Further, $\mathrm{var}(F^{n,p}) = \mathbb{E}[(F_{n,p})^2] - \mathbb{E}[F^{n,p}]^2$, from (25),

$$\mathrm{var}(F^{n,p}) = \frac{p(1-p)(3-p)}{4}\left(1-\frac{p}{2}\right)^{2n-1} - \frac{p(1-p)}{4}\left(\frac{1-\frac{p}{2}}{2}\right)^{n+1}.$$

□

Received August 2024; revised September 2024; accepted October 2024