Asynchronous Distributed Algorithms for Average Consensus Problems

Anja Kroon
Signal Processing Systems
Delft University of Technology
Delft, Netherlands

Frank Harraway
Signal Processing Systems
Delft University of Technology
Delft, Netherlands

I. Introduction

Reaching consensus within extensive distributed networks poses a significant fundamental challenge across numerous disciplines [1]. Traditional methods often rely on centralized nodes that maintain a global state and coordinate activities across the network. However, this centralized approach introduces a central point of failure. Centralized systems further struggle with issues of privacy, as all information must be routed through and often stored by a single entity. In practical applications, it is difficult to synchronously update all nodes in a network. Synchronous algorithms are sensitive to changes in network topology thereby decreasing their robustness in real-world applications. In response to these challenges, there is a growing interest in asynchronous, decentralized and distributed algorithms that operate effectively without the need for a central coordinator. This paper will examine various asynchronous distributed algorithms and their performance in terms of convergence rate and robustness to changes in network topology.

The problem setting of interest is detailed in section II. Various asynchronous distributed algorithms are discussed in detail in section III. The evaluation methods and experimental setup also follow. The results of the experiments are then presented in section IV followed by a discussion in subsection IV-F [1].

II. Problem Setting

A toy problem setting is considered where, in a plant, temperature sensors are randomly placed in a square area of 100 x 100 m^2. The objective of this average consensus problem is for all sensors in the plant to obtain the average of their temperatures as depicted in Figure 1. It is assumed that the sensor values are fixed during the convergence time of the network. The nodes are connected in a scheme which forms a random geometric graph (RGG). An RGG is constructed by connecting every node to all other nodes within a specified radius. A RGG is connected with probability $1 - 1/n^2$ if the radius is $r = \sqrt{2log(n)/n}$ where n is the number of nodes in the RGG and a probability of 1 indicates it is guaranteed to be connected. With this scheme, the sensors adequately cover the area and form a connected RGG with high probability.

[1]The full codebase for the algorithms and experiments in this report can be found at Link.

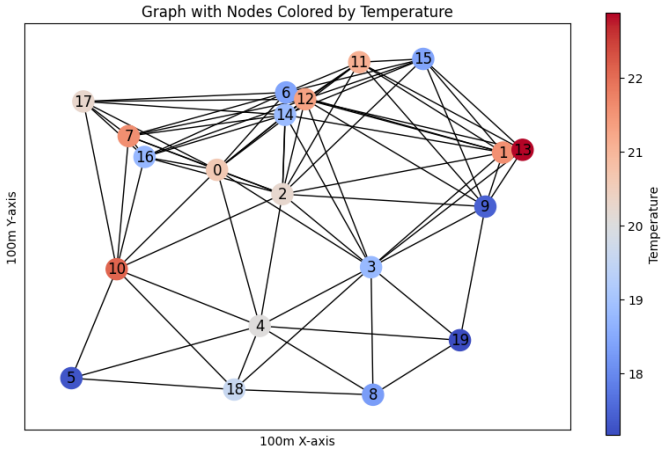

Fig. 1: A random geometric graph (RGG) colour by the node temperature values is generated with 20 nodes. The graph is connected with probability 99.75%.

While the temperature setup may seem trivial, this toy setting can be extended to the real-world by considering a flock of drones flying together. Each drone may be affected by the wind or drag of the drones surrounding it. To dynamically adjust its own speed to match the speed of the flock, the drones may solve an average consensus problem in a distributed asynchronous manner to remain flying as a flock.

III. Methodology

To achieve a consensus over the network, three asynchronous algorithms of interest are explored: the asynchronous distributed averaging algorithm (DA), the random gossip algorithm (RG) and the Primal-Dual Method of Multipliers (PDMM). The random gossip algorithm is inherently asynchronous, whereas an adapted asynchronous implementation of DA and PDMM are used. These algorithms are explained in subsection III-A, subsection III-B and subsection III-C, respectively, and the method of assessment to compare the algorithms is identified in subsection III-D. The python experiments and implementation can be found on Github [2].

A. Distributed Averaging (Asynchronous)

As a precursor to randomized gossip, the asynchronous distributed averaging algorithm is introduced. In this algorithm,

a random node i from the network is selected and an average is computed among its neighbors, nodes $j \in \mathcal{N}(i)$. The selected node and its neighbors then update their values based on the group average. This approach does not depend on global knowledge of the network. During the iterations, only node i and its neighbors are updated affirming the algorithm is asynchronous. The number of transmissions for each iteration is equal to twice the number of neighbors of node i. Node i first obtains the temperature values from the neighbors and then distributes the calculated average back to its neighbors. This algorithm is known, however, to be sensitive to changes in the network topology [3]. Therefore, randomized gossip is considered as an improvement which is inherently more robust to network topology changes.

B. Randomized Gossip

In the randomized gossip algorithm, an RGG is defined, $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, and node $i \in \mathcal{V}$ is uniformly selected at random. This means, in a network with three nodes, each node has a 1/3 probability of being selected. Then, a neighbor node j of node i is also randomly selected at uniform. The randomized gossip algorithm then averages the temperature values, x_i and x_j, and updates the temperature values stored at that node. Recall, the objective is to find the distributed average across all sensors in the network. This process is repeated until the stopping criterion that the $error = \|\mathbf{x} - x_{ave}\mathbf{1}\|_2^2$ is met. As this algorithm updates only some nodes in the network at a given time, it is considered to inherently be an asynchronous algorithm. Each iteration produces two transmissions across the link between node i and j. The first is from node i retrieving the temperature value from node j. The second is from node i sending node j an updated temperature value. [4] Selecting neighbors uniformly at random in the randomized gossip scheme is called the natural averaging algorithm. It is comparable to the performance of the optimal algorithm for random geometric graphs. [3] The algorithm, 1, is applied to the problem setup to solve the temperature average consensus problem.

Algorithm 1 Random Gossip

1: **Initialize:** $\mathbf{x}^{(0)} = \mathbf{a}$ (sensor values), ϵ (threshold)
2: **Error Metric:** $error = \|\mathbf{x} - x_{ave}\mathbf{1}\|_2^2$
3: **while** $error > \epsilon$ **do**
4: Uniformly select a random node i
5: Uniformly select a neighbor of node i, denote node j
6: **Averaging:**
7: $x_i^{(k+1)} = (\, x_i^{(k)} + \, x_j^{(k)})/2$
8: $x_j^{(k+1)} = (\, x_i^{(k)} + \, x_j^{(k)})/2$
9: Transmissions += 1
10: **end while**

C. Asynchronous Primal-Dual Method of Multipliers (PDMM)

The unicast asynchronous Primal-Dual Method of Multipliers (PDMM) algorithm is implemented as the second asynchronous distributed algorithm [5]. Unicast PDMM only updates node variables associated with a single node at a time. The asynchronous PDMM can be used to find the solution $\mathbf{x}$ (temperatures) to the consensus problem using the measured temperatures $\mathbf{a}$ and the constraint that all neighbors of each node must be equal, $x_i - x_j = 0$. The consensus problem can be written as,

$$\begin{aligned} \underset{x}{\text{minimize}} \quad & f(x) = \sum_{i=1}^{n} \frac{1}{2}(x_i - a_i)^2 \\ \text{subject to} \quad & x_i - x_j = 0, \quad \forall \quad (i,j) \in E \end{aligned} \tag{1}$$

where x_i denotes the current temperature value at node i, a_i denotes the original sensor measurement at node i, and n details the number of nodes in the network. PDMM finds the solution by solving the dual problem of Equation 1. When representing the dual problem in terms of the conjugate function it becomes clear that the conjugate functions related to each edge are dependent on the edge and not just the node due to the dual variable, v_{ij}. Therefore, the single constraint in the dual problem is handled by introducing two auxiliary $\lambda_{i|j}, \lambda_{j|i} \in \mathbb{R}^m$ for each of the edges $\mathcal{E}$ in the graph. This is necessary to ensure that the problem is separable and can be solved in a distributed and asynchronous manner. The newly introduced variables are constrained to be equivalent to the original dual variable, v_{ij}. Using Peaceman-Rachford splitting to efficiently and iteratively maximize the dual problem, the $\mathbf{z}^{(k)}$ and the $\boldsymbol{\lambda}^{(k)}$ iterates are determined where $\mathbf{z}$ is an auxiliary variable introduced by the splitting algorithm. Now, $\boldsymbol{\lambda}^{(k)}$ can be determined in terms of the resolvent of $\mathbf{z}$. To obtain the desired $\mathbf{x}^{(k)}$ iterate, the inverse relation of conjugate subdifferentials and subdifferentials is used. Since the original dual variable v_{ij} has been split into two separate variables it is necessary to find the projection $\mathbf{y}$ of $\mathbf{z}$ back onto the feasible set.

Using the process explained above the Equation 1 problem is separable over the nodes in the dual domain allowing for the PDMM algorithm to be implemented in a distributed asynchronous manner as shown in algorithm 2. The constraint can be rewritten as $\mathbf{A}\mathbf{x} = 0$ where $\mathbf{A}$ is a $m \times n$ matrix. $\mathbf{A}$ is defined as $A_{ij} = 1$ if $i < j$ and $A_{ij} = -1$ where $(i,j) \in \mathcal{E}$.

The asynchronous PDMM algorithm 2 uniformly randomly selects the node i and performs the primal update by minimising the node value x_i for the PDMM consensus problem using information from its neighbours (where d_i is the degree of node i) [3]. Thereafter the dual update is performed updating $y_{i|j}$. The updated dual variables are then made available to their neighbours for the next iteration (in code simulation on a centralised system no step needs to be performed). Finally, the auxiliary variables are updated using neighbouring dual variables. This is repeated until the stopping criterion, defined the same as in RG ($||\mathbf{x}_k - \mathbf{x}_{avg}||_2^2 < \epsilon$), is met. Transmissions are counted for each update of the primal variable x_i.

D. Method of Assessment

The algorithmic performance is first examined under ideal scenarios where the network (a RGG) does not change. Then, disturbances to the network are introduced including transmission failures, dropping nodes, and adding nodes which better simulate real-world scenarios.

Algorithm 2 Asynchronous PDMM

Inputs: $\mathbf{A} \in \mathbb{R}^{m \times n}$, $\mathbf{a} \in \mathbb{R}^n$(sensor values), ϵ (threshold), c (convergence rate parameter)
Initialise: $\mathbf{z}^{(0)}, \mathbf{y}^{(0)} = \mathbf{0}^{2m}$, $\mathbf{x}^{(0)} = \mathbf{0}^n$,
Error Metric: $error = \|\mathbf{x} - x_{ave}\mathbf{1}\|_2^2$
while $error > \epsilon$ **do**
    Uniformly select a random node i
    $x_i^{(k+1)} = \frac{a_i - \Sigma_{j \in \mathcal{N}(i)} A_{ij} z_{i|j}}{1 + cd_i}$
    for all $j \in \mathcal{N}(i)$ **do**
        $y_{i|j}^{(k+1)} = z_{i|j}^{(k)} + 2c\left(A_{ij} x_i^{(k+1)}\right)$
    end for
    for all $j \in \mathcal{N}(i)$ **do**
        $\text{node}_j \leftarrow \text{node}_i\left(y_{i|j}^{(k+1)}\right)$
    end for
    for all $j \in \mathcal{N}(i)$ **do**
        $z_{i|j}^{(k+1)} = y_{j|i}^{(k+1)}$
    end for
    Transmissions += 1
end while

Transmission failures (TF) occur when an intended communication between two nodes does not occur. To simulate transmission failures, the intended communications over links in the algorithms are randomly skipped by the desired transmission failure rate. For example, if an algorithm performs 10 transmissions in an ideal case without TF, it would execute 5 transmissions for a case of 50% TF.

The dropping of nodes is another failure which can occur through an adversarial attack or through accidental disconnection of a node in the network. Two types of node droppings are considered – bulk and sequential. In bulk dropping, a large amount of nodes are dropped from the existing network at a specific moment in time. In the real world, this may occur if an adversary uses a jammer to disconnect a group of nodes from the network. In sequential dropping, a singular node is dropped from the network at a specific moment. This process is repeated until the desired amount of nodes are dropped. This procedure may occur realistically when faulty nodes drop out of the network due to age, for example. Nodes may be added to the network maliciously or in an attempt to perform network recovery after an adversarial attack.

When the nodes are introduced or removed from the network the average of the nodes is recalculated as the true mean across the nodes changes. As nodes are introduced, they are added with the same connecting radius r as the original graph.

The algorithms under ideal and network topology changes are chiefly examined on their convergence rate which is measured by error (how closely the sensor values match the true average) versus the number of transmissions over the same network.

E. Experimental Setup

To ensure a fair comparison of algorithms, the number of nodes in all results is fixed to be $n = 200$. All networks created and modified maintain the random geometric network structure. The temperature measurements at each node are generated following a gaussian distribution, $t \sim \mathcal{N}(22, 5)$. These measurements are representative of real data given the problem setting of measuring temperature across a 100 m^2 area. The stopping criterion for all algorithms is set to $||\mathbf{x}_k - \mathbf{x}_{avg}||_2^2 < \epsilon$ where $\epsilon = 10^{-12}$. The original measurements of each sensor are assumed to be constant for the time needed for consensus algorithm convergence.

IV. Results

Three algorithms are considered for their ability to solve the distributed average consensus problem in an asynchronous manner: distributed averaging (DA), randomized gossip (RG), and primal-dual method of multipliers (PDMM). Tuning of the variable c is first reported for the PDMM algorithm. The asynchronous algorithms' performance in solving the average consensus problem is considered under ideal conditions. Then, the algorithms are considered under transmission failures and changes to the network topology to simulate performance in real-world settings.

A. Choice of Hyperparameter c in PDMM

PDMM includes a hyperparameter, c, which can be tuned to the problem setting. If the problem converges for a c, it will converge for any c. However, the rate of convergence can be improved by fine-tuning c. Therefore, c values ranging from $\{0.1, 0.9\}$ are considered. Based on empirical results shown in Appendix , the lowest number of transmissions is attained with a value of c = 0.4.

B. Under Ideal Conditions

Under ideal conditions, where there are no communication failures or changes to network topology, DA, RG, and PDMM converge to a solution, seen in Figure 2. PDMM converges in the least number of transmissions followed closely by the distributed averaging algorithm. Random gossip converges the slowest requiring more than 2x transmissions than PDMM. Notably, all algorithms exhibit a linear convergence rate on a log scale, a promising result for real-world applications.

C. Transmission Failures (TF)

Moving away from the ideal scenario, it is important to consider transmission failures (TF) which are more likely to occur in a real-world setting. Algorithm performance under 0, 25, 50, and 75 % transmission failure is examined in Figure 3, Figure 4 and Figure 5, respectively.

For various transmission failure rates, the DA algorithm is unable to reach convergence. These results support the known behavior of DA to be sensitive to changes in the network topology. When a transmission is not sent, the nodes update independently of the node omitted from the transmission (depending on the type of transmission failure), leading to the nodes averaging to the incorrect average as observed in Figure 3.

In randomized gossip, TFs occur when node i is unable to contact its randomly selected neighbor node j or when

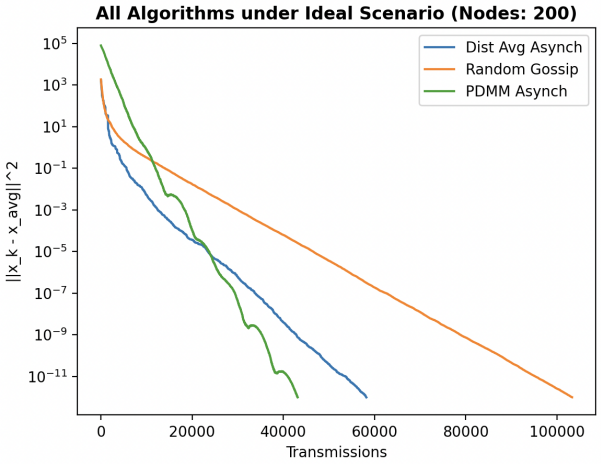

Fig. 2: Performance comparison under Ideal Conditions. PDMM converges first. Random gossip converges last.

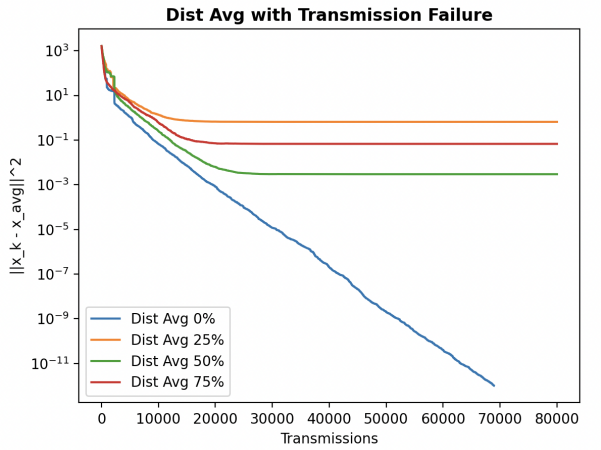

Fig. 3: Transmission failures for DA. System unable to converge to the true error with transmission failures.

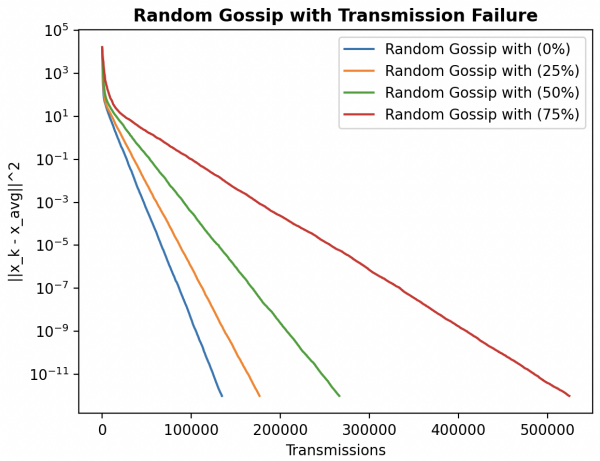

Fig. 4: Transmission failures for RG. Convergence remains albeit at a slower rate.

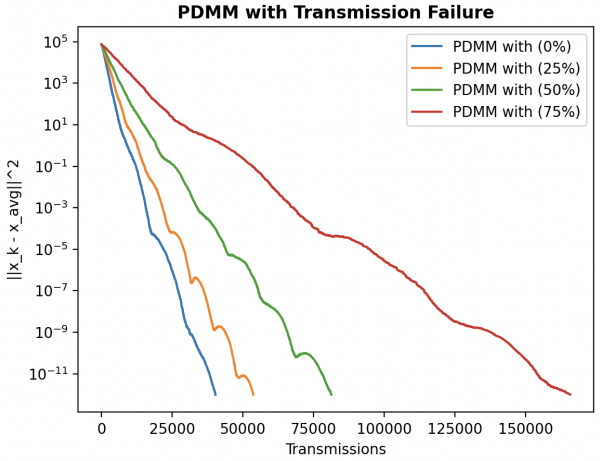

Fig. 5: Transmission failures for PDMM. Convergence remains albeit at a slower rate.

node j is unable to share its current temperature value. There is only one active link per iteration so a failure of that link effectively results in a wasted iteration. TF does, therefore, result in a slow-down in algorithmic convergence. Notably, the increase in the number of transmissions needed to converge from 25% to 50% TF is approximately double the amount of transmissions needed to converge from 0% to 25% TF. This behavior continues as the TF rate increases to 75%. The convergence rate remains linear on the log-scale despite the TFs.

In PDMM, similar behavior is noted. The convergence rate decreases as the number of TF increases. Further, as in RG, the increase in the number of transmissions needed to converge from 25% to 50% TF is approximately double the amount of transmissions needed to converge from 0% to 25% TF. The convergence still remains linear on a log scale. Both RG and PDMM are robust to TF in the sense that they still converge albeit at a slower rate.

The number of transmissions required increases as expected for both RG and asynchronous PDMM. In both Figure 5 and Figure 4 it can be said that the transmission failures correspond to Equation 2

$$\#\text{Transmissions with X\%TF} = \frac{\#\text{Transmissions for 0\% TF}}{100\% - \text{X\% TF}} \quad (2)$$

where the expected transmissions are recalculated as a ratio of the number of transmission failures. As an example in Figure 4 the number of transmissions for 75% transmission failure is $\frac{130000}{1-0.75} = 520000$ transmissions.

D. Removal of Nodes

In addition to transmission failures, the network topology can be modified by the removal of nodes in the network. Nodes may be removed in the network in real settings due to complete node failure. There are two types of node drops: bulk removal (presented here) and sequential removal (Figure A). In a network of 200 nodes, the bulk removal procedure includes dropping 100 nodes or 50% of the nodes. In the implementation

of bulk removal, all sensor values remain fixed at their last known updated values. The results on RG and PDMM presented in Figure 6 demonstrate that the algorithms still converge linearly on a log-scale but slower than before the node drops.

For random gossip, removing nodes from the network generally results in an improved convergence rate. Each iteration in random gossip involves only two nodes. Therefore, removing nodes typically has little impact on the calculations of each iteration and simply reduces the overall problem complexity. Seen in Figure 6, there is not necessarily a spike in the error when nodes are removed. This may occur if removing nodes moves the system closer to the final solution (the true average of all the initial sensor values).

For PDMM, convergence slows after the removal of nodes because as the nodes are removed from the graph the number of neighbours per node becomes fewer. PDMM is highly dependent on neighbour nodes and performs better on a more highly connected graph. However, the algorithm is still able to converge. Because the sensor values remain fixed at their last known values, the error spikes upon removal but only to a fraction of the initial error. The algorithm continues to converge to the optimal solution after the spike suggesting it is indeed robust to node removals.

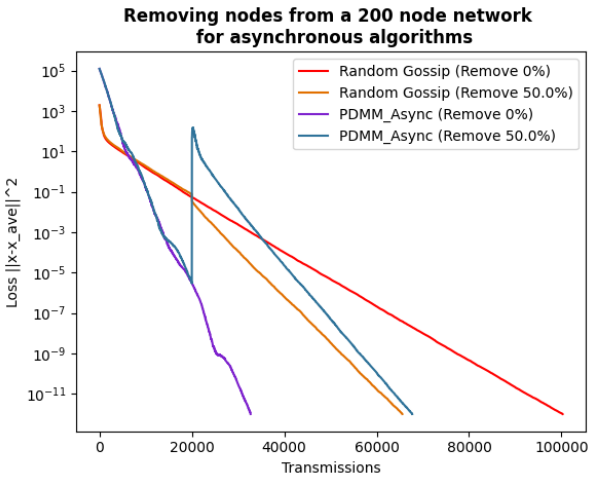

Fig. 6: Removing nodes in bulk for RG and PDMM.

E. Bulk Adding of Nodes

After a network has been depleted due to node drops, it is typical to rebuild the network by adding nodes. Similarly to node removal, nodes can be added in bulk (considered here) or in sequence (Figure A). For a network of 200 nodes, an adding rate of 50% is considered to add 100 new nodes to the network on both algorithms Figure 7. These nodes contain measurements following the same distribution as the original nodes and are connected in a random geometric manner to the existing nodes.

Adding nodes to the network generally worsened the convergence rate of the random gossip algorithm. For a significant addition of 50 %, the convergence rate moderately worsens. The error did spike upon adding new nodes to the network. The spike increased to a magnitude comparable to the initial error. This was despite fixing the current temperature values of the 200 nodes before the bulk add. The high spike and linear convergence behavior after bulk addition of nodes in Figure 7 suggest that the RG algorithm can converge but essentially undergoes a "restart" when nodes are added in bulk.

For PDMM, adding nodes to the network in bulk similarly worsened the convergence rate seen in Figure 7. As nodes are added to the network, the number of neighbors of node i will correspondingly increase resulting in increased problem complexity. Further, the error spike also increased akin in magnitude to the error of the first iteration. This spike is likely larger than the RG spike due to the initialization of variables $\mathbf{z}, \mathbf{y} = \mathbf{0}$. This indicates the algorithm "restarts" upon adding 50% nodes in bulk. Despite adding many nodes, the PDMM algorithm can converge at only a marginally worse rate marginally than the algorithm in the ideal scenario. Adding more nodes at the same radius of connection as the original graph creates a more connected graph which is why the change in convergence rate is marginal.

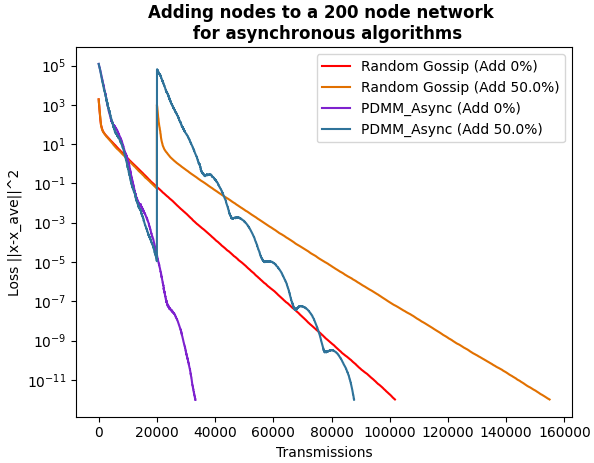

Fig. 7: Adding nodes in bulk for RG and PDMM.

F. Discussion & Conclusion

The asynchronous RG and PDMM algorithms are easily able to solve the average consensus problem under ideal conditions. However, the advantages of particular algorithms become apparent when considering three types of network topology changes – transmission failures, dropping nodes, and adding nodes. RG is inherently robust to network changes due to its iterations only contacting a single neighbor. However, only contacting a single neighbor results in a slower convergence rate. The RG algorithm can, however, be improved by including prior information about sensor locations, states of neighboring nodes, or various broadcast protocols. Alternatively, a different approach to solving the average consensus problem can be considered – PDMM. PDMM is similarly robust to network changes and converges in fewer transmissions. In the ideal case, PDMM converges faster than the original distributed averaging

algorithm. Therefore, PDMM becomes an improved choice over RG when considering performance in the ideal setting and under network topology changes. Distributed averaging converged quickly for the ideal case but was sensitive to changes in the network topology and failed under real-world scenarios.

Three types of asynchronous algorithms are investigated under ideal conditions and under network topology changes. While RG and PDMM demonstrated good robustness overall, PDMM provided a faster convergence rate in both the ideal condition and under the three types of network topology changes.

Future work may include how initialising PDMM auxiliary and dual variables impact convergence time. Regarding RG, future considerations can be placed upon incorporating prior information to attain faster convergence as the geographic gossip algorithm does, for example. Future experimentation should be conducted to implement this algorithm in a real-world setting where data is further from the ideal setting and the number of transmissions and communication protocols may play a larger role in algorithm convergence. It may lastly be interesting to explore how these consensus algorithms extend to real-world settings where the temperature values at each node change with time. Particularly, it may be intriguing to consider how the previous solution to the average consensus problem can be incorporated as a prior into the next average consensus iteration and the corresponding impact on convergence time.

REFERENCES

[1] L. Lamport, "The byzantine generals problem." https://lamport.azurewebsites.net/pubs/byz.pdf, 1982. Accessed: 01/04/2024.

[2] F. Harraway and A. Kroon, "Github distributedsignalproc." https://github.com/AnjaKroon/DistributedSignalProc, 2024.

[3] R. Heusdens, "Lecture notes in distributed signal processing," 2024.

[4] S. Boyd, A. Ghosh, B. Prabhakar, and D. Shah, "Randomized gossip algorithms," *IEEE Transactions on Information Theory*, 2006.

[5] T. W. Sherson, R. Heusdens, and W. B. Kleijn, "Derivation and analysis of the primal-dual method of multipliers based on monotone operator theory," *IEEE Transactions on Signal and Information Processing over Networks*, 2019.

APPENDIX A
HYPERPARAMETER TUNING FOR c IN PDMM

The asynchronous PDMM algorithm was run with various values for c. From these experiments, **the optimal value of c is chosen to be** 0.4. The plot corresponding to this claim in the paper can be found in Figure 10

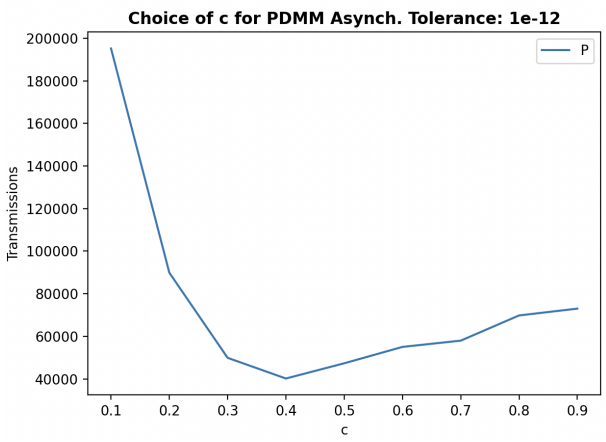

Fig. 8: C choice asynchronous PDMM, N=200

APPENDIX B
SEQUENTIAL REMOVAL/ADDING OF NODES AND RESULTS

In the report, sequential node removal and adding are mentioned. The results corresponding to those sequential adding and removal scenarios can be found below. Based on these experiments, **it is best to add and remove nodes in bulk as the algorithm is essentially "stalled" from converging until all nodes have been added or removed**. Best is defined here in terms of convergence. Therefore, bulk add/removal is used in the paper as a discussion point.

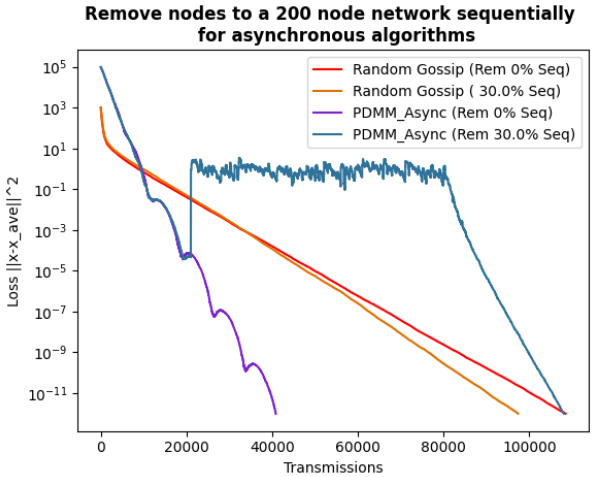

Fig. 9: Sequential removal of nodes, N=200

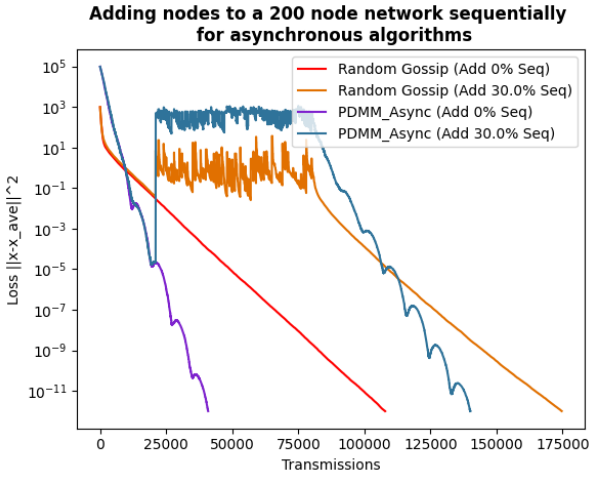

Fig. 10: Sequential adding of nodes, N=200

APPENDIX C
TESTING VARIOUS IMPLEMENTATIONS

There are numerous approaches to implementing the asynch. distributed averaging, random gossip, and asynch. PDMM algorithms. The chosen approach for all algorithms is based on the node i and node j interpretation. However, it is possible to implement many of these algorithms using matrices. In particular, we explored whether implementing the DA algorithm with the W matrix as defined in [3] would impact the convergence time. According to the results presented in Figure 11, there exists negligible distinction due to randomness. These results demonstrate that **it is likely that various methods of algorithm implementation will have a marginal impact on the final results**.

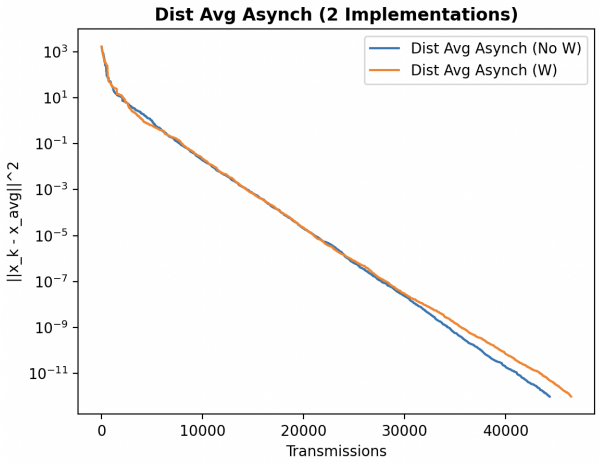

Fig. 11: Two implementations of the distributed asynchronous algorithm have negligible impact on algorithm convergence.

APPENDIX D
RUN TIME OF ASYNCHRONOUS ALGORITHMS FOR DIFFERENT NUMBER OF NODES IN THE NETWORK

The run time of the various algorithms are presented in Table I under ideal conditions (no TF or adding/dropping of

TABLE I: Algorithm Run Times

Number of Nodes	PDMM [sec]	RG [sec]	DA [sec]
50	0.587	0.209	0.008
100	1.341	4.686	0.084
200	4.630	17.947	0.143
300	7.785	94.405	0.257

nodes). As the number of nodes increase, the run time increases as anticipated. The asynchronous distributed averaging (DA) algorithm always converges the in the least number of iterations under idael conditions. PDMM follows DA's performance providing a moderate convergence time of approx. 8 seconds for 300 nodes. **RG evidently struggles as the number of nodes increases** requiring nearly 100 seconds to converge with 300 nodes. Although DA's results are impressive, **DA only converges under ideal conditions** proving a steep limitation to the practicality of the algorithm. **PDMM is therefore a good alternative providing robustness with reasonable convergence times.** These convergence times are highly dependent, however, on the specific algorithmic implementation and should therefore be treated as a rough estimate of performance rather than a given.