Journal of Machine Learning Research 21 (2020) 1-47 Submitted 12/18; Revised 2/20; Published 3/20

Robust Asynchronous Stochastic Gradient-Push: Asymptotically Optimal and Network-Independent Performance for Strongly Convex Functions

Artin Spiridonoff ARTIN@BU.EDU
Alex Olshevsky ALEXOLS@BU.EDU
Ioannis Ch. Paschalidis YANNISP@BU.EDU
Division of Systems Engineering
Boston University
Boston, MA 02215, USA

Editor: Suvrit Sra

Abstract

We consider the standard model of distributed optimization of a sum of functions $F(\mathbf{z}) = \sum_{i=1}^{n} f_i(\mathbf{z})$, where node i in a network holds the function $f_i(\mathbf{z})$. We allow for a harsh network model characterized by asynchronous updates, message delays, unpredictable message losses, and directed communication among nodes. In this setting, we analyze a modification of the Gradient-Push method for distributed optimization, assuming that (i) node i is capable of generating gradients of its function $f_i(\mathbf{z})$ corrupted by zero-mean bounded–support additive noise at each step, (ii) $F(\mathbf{z})$ is strongly convex, and (iii) each $f_i(\mathbf{z})$ has Lipschitz gradients. We show that our proposed method asymptotically performs as well as the best bounds on centralized gradient descent that takes steps in the direction of the sum of the noisy gradients of all the functions $f_1(\mathbf{z}), \ldots, f_n(\mathbf{z})$ at each step.

Keywords: distributed optimization, stochastic gradient descent.

1. Introduction

Distributed systems have attracted much attention in recent years due to their many applications such as large scale machine learning (e.g., in the healthcare domain, Brisimi et al., 2018), control (e.g., maneuvering of autonomous vehicles, Peng et al., 2017), sensor networks (e.g., coverage control, He et al., 2015) and advantages over centralized systems, such as scalability and robustness to faults. In a network comprised of multiple agents (e.g., data centers, sensors, vehicles, smart phones, or various IoT devices) engaged in data collection, it is sometimes impractical to collect all the information in one place. Consequently, distributed optimization techniques are currently being explored for potential use in a variety of estimation and learning problems over networks.

This paper considers the separable optimization problem

$$\min_{\mathbf{z} \in \mathbb{R}^d} F(\mathbf{z}) := \sum_{i=1}^{n} f_i(\mathbf{z}), \tag{1}$$

where the function $f_i : \mathbb{R}^d \to \mathbb{R}$ is held only by agent i in the network. We assume the agents communicate through a directed communication network, with each agent able to send messages to its out-neighbors. The agents seek to collaboratively agree on a minimizer to the global function $F(\mathbf{z})$.

This fairly simple problem formulation is capable of capturing a variety of scenarios in estimation and learning. Informally, $\mathbf{z}$ is often taken to parameterize a model, and $f_i(\mathbf{z})$ is a loss function measuring how well $\mathbf{z}$ matches the data held by agent i. Agreeing on a minimizer of $F(\mathbf{z})$ means agreeing on a model that best explains all the data throughout the network – and the challenge is to do this in a distributed manner, avoiding techniques such as flooding which requires every node to learn and store all the data throughout the network. For more details, we refer the reader to the recent survey by Nedic et al. (2018).

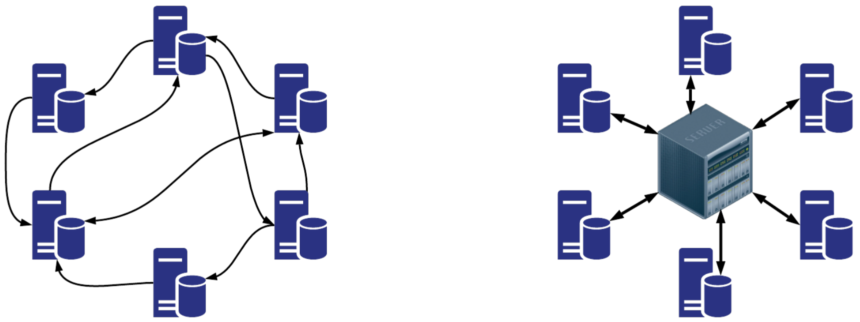
In this work, we will consider a fairly harsh network environment, including message losses, delays, asynchronous updates, and directed communication. The function $F(\mathbf{z})$ will be assumed to be strongly convex with the individual functions $f_i(\mathbf{z})$ having a Lipschitz continuous gradient. We will also assume that, at every time step, node i can obtain a noisy gradient of its function $f_i(\mathbf{z})$. Our goal will be to investigate to what extent distributed methods can remain competitive with their centralized counterparts in spite of these obstacles.

1.1. Literature Review

Research on models of distributed optimization dates back to the 1980s, see Tsitsiklis et al. (1986). The separable model of (1) was first formally analyzed in Nedic and Ozdaglar (2009), where performance guarantees on a fixed-stepsize subgradient method were obtained. The literature on the subject has exploded since, and we review here only the papers closely related to our work. We begin by discussing works that have focused on the effect of harsh network conditions.

A number of recent papers have studied asynchronicity in the context of distributed optimization. It has been noted that asynchronous algorithms are often preferred to synchronous ones, due to the difficulty of perfectly coordinating all the agents in the network, e.g., due to clock drift. Papers by Recht et al. (2011); Li et al. (2014); Agarwal and Duchi (2011); Lian et al. (2015) and Feyzmahdavian et al. (2016) study asynchronous parallel optimization methods in which different processors have access to a shared memory or parameter server. Recht et al. (2011) present a scheme called HOGWILD!, in which processors have access to the same shared memory with the possibility of overwriting each other's work. Li et al. (2014) proposes a parameter server framework for distributed machine learning. Agarwal and Duchi (2011) analyze the convergence of gradient-based optimization algorithms whose updates depend on delayed stochastic gradient information due to asynchrony. Lian et al. (2015) improve on the earlier work by Agarwal and Duchi (2011), and study two asynchronous parallel implementations of Stochastic Gradient (SG) for nonconvex optimization; establishing an $\mathcal{O}_k(1/\sqrt{k})$ convergence rate for both algorithms. Feyzmahdavian et al. (2016) propose an asynchronous mini-batch algorithm that eliminates idle waiting and allows workers to run at their maximal update rates.

The works mentioned above consider a *centralized* network topology, i.e., there is a central node (parameter server or shared memory) connected to all the other nodes. On the other hand, in a *decentralized* setting, nodes communicate with each other over a connected network without depending on a central node (see Figure 1). This setting reduces the



(a) Decentralized network topology.

(b) Centralized network topology.

Figure 1: Different network topologies.

communication load on the central node, is not vulnerable to failures of that node, and is more easily scalable.

For analysis of how decentralized asynchronous methods perform we refer the reader to Mansoori and Wei (2017); Tsitsiklis et al. (1986); Srivastava and Nedic (2011); Assran and Rabbat (2018); Nedic (2011); Wu et al. (2018) and Tian et al. (2018). We note that of these works only Tian et al. (2018) is able to obtain an algorithm which agrees on a global minimizer of (1) with non-random asynchronicity, under the assumptions of strong convexity, noiseless gradients and possible delays. On the other hand, the papers Nedic (2011) and Wu et al. (2018) obtain convergence in this situation under assumptions of natural randomness in the algorithm: the former assumes randomly failing links while the latter assumes that nodes make updates in random order.

The study of distributed separable optimization over directed graphs was initiated in Tsianos et al. (2012b), where a distributed approach based on dual averaging with convex functions over a fixed graph was proposed and shown to converge at an $\mathcal{O}_k(1/\sqrt{k})$ rate. Some numerical results for such methods were reported in Tsianos et al. (2012a). In Nedic and Olshevsky (2015), a method based on plain gradient descent converging at a rate of $\mathcal{O}_k((\ln k)/\sqrt{k})$ was proposed over time-varying graphs. This was improved in Nedic and Olshevsky (2016) to $\mathcal{O}_k((\ln k)/k)$ for strongly convex functions with noisy gradient samples. More recent works on optimization over directed graphs are Akbari et al. (2017), which considered online convex optimization in this setting, and Assran and Rabbat (2018), which considered combining directed graphs with delays and asynchronicity. The main tool for distributed optimization is the so-called "push sum" method introduced in Kempe et al. (2003), which is widely used to design communication and optimization schemes over directed graphs. More recent references are Bénézit et al. (2010); Hadjicostis et al. (2016), which provide a more modern and general analysis of this method, and the most comprehensive reference on the subject is the recent monograph by Hadjicostis et al. (2018). We also mention Xi and Khan (2017a); Xi et al. (2018); Nedic et al. (2017), where an approach based on push-sum was explored. A parallel line of work in this setting based on the the ADMM model, where updates are allowed to include a local minimization step, was explored in Brisimi et al. (2018); Chang et al. (2016a,b) and Hong (2017).

The reason directed graphs present a problem is because much of distributed optimization relies on the primitive of "multiplication by a doubly stochastic matrix:" given that each node of a network holds a number x_i, the network needs to compute y_i, where $\mathbf{x} = (x_1, \ldots, x_n)^\top$, $\mathbf{y} = (y_1, \ldots, y_n)^\top$ and $\mathbf{y} = \mathbf{W}\mathbf{x}$ for some doubly stochastic matrix $\mathbf{W}$ with positive spectral gap. This is pretty easy to accomplish over undirected graphs (see Nedic et al., 2018) but not immediate over directed graphs. A parallel line of research focuses on distributed methods for constructing such doubly stochastic matrices over directed graphs – we refer the reader to Dominguez-Garcia and Hadjicostis (2013); Gharesifard and Cortés (2012); Domínguez-García and Hadjicostis (2014). Unfortunately, to the authors' best knowledge, no explicit and favorable convergence time guarantees are known for this procedure. Another line of work (Xi and Khan, 2017b) takes a similar approach, based on construction of a doubly stochastic matrix with positive spectral gap after the introduction of auxiliary states. Among works with undirected graphs, Scaman et al. (2017) derived the optimal convergence rates for smooth and strongly convex functions and introduced the multi-step dual accelerated (MSDA) algorithm with optimal linear convergence rate in the deterministic case.

Dealing with message losses has always been a challenging problem for multi-agent optimization protocols. Recently, Hadjicostis et al. (2016) resolved this issue rather elegantly for the problem of distributed average computation by having nodes exchange certain running sums. It was shown in Hadjicostis et al. (2016) that the introduction of these running sums is equivalent to a lossless algorithm on a slightly modified graph. We also refer the reader to the follow-up papers by Su and Vaidya (2016b,a, 2017). We will use the same approach in this work to deal with message losses.

In many applications, calculating the exact gradients can be computationally very expensive or impossible Lan et al. (2018). In one possible scenario, nodes are sensors that collect measurements at every step, which naturally corrupts all the data with noise. Alternatively, communication between agents may insert noise into information transmitted between them. Finally, when $f_i(\mathbf{z})$ measures the fit of a model parameterized by the vector $\mathbf{z}$ to the data of agent i, it may be efficient for agent i to randomly select a subset of its data and compute an estimate of the gradient based on only those data points (Alpcan and Bauckhage, 2009). Motivated by these considerations, a literature has arisen studying the effects of stochasticity in the gradients. For example, Srivastava and Nedic (2011) showed convergence of an asynchronous algorithm for constrained distributed stochastic optimization, under the presence of local noisy communication in a random communication network. In Pu and Nedic (2018), two distributed stochastic gradient methods were introduced, and their convergence to a neighborhood of the global minimum (under constant step-size) and to the global minimum (under diminishing stepsize) was analyzed. In work by Sirb and Ye (2016), convergence of asynchronous decentralized optimization using delayed stochastic gradients has been shown.

The algorithms we will study here for stochastic gradient descent are based on the standard "consensus+gradient descent" framework: nodes will take steps in the direction of their gradients and then "reconcile" these steps by moving in the directions of an average of their neighbors in the graph. We refer the reader to Nedic et al. (2018); Yuan et al. (2016), for a more recent and simplified analysis of such methods. It is also possible to take a more modern approach, pioneered in Shi et al. (2015), of using the past history to make updates; such schemes have been shown to achieve superior performance in recent years (see Shi et al., 2015; Sun et al., 2016; Oreshkin et al., 2010; Nedic et al., 2017; Xi and Khan, 2017a; Xi et al., 2018; Qu and Li, 2017; Xu et al., 2015; Qu and Li, 2019; Di Lorenzo and Scutari, 2016); we refer the reader to Pu and Nedic (2018) which took this approach.

One of our main concerns in this paper is to develop decentralized optimization methods which perform as well as their centralized counterparts. Specifically, we will compare the performance of a distributed method for (1) on a network of n nodes with the performance of a centralized method which, at every step, can query all n gradients of the functions $f_1(\mathbf{z}), \ldots, f_n(\mathbf{z})$. Since the distributed algorithm gets noise-corrupted gradients, so should the centralized method. Thus, the natural approach is to compare the distributed method to centralized gradient descent which moves in the direction of the sum of the gradients of $f_1(\mathbf{z}), \ldots, f_n(\mathbf{z})$. This method of comparison keeps the "computational power" of the two nodes identical.

Traditionally, the bounds derived on distributed methods were considerably worse than those derived for centralized methods. For example, the papers by Nedic and Olshevsky (2015, 2016) had bounds for distributed optimization over directed graphs that were worse than the comparable centralized method (in terms of rate of error decay) by a multiplicative factor that, in the worst case, could be as large as $n^{\mathcal{O}(n)}$. This is typical over directed graphs, though better results are possible over undirected graphs. For example, in Olshevsky (2017), in the model of noiseless, undelayed, synchronous communication over an undirected graph, a distributed subgradient method was proposed whose performance, relative to a centralized method with the same computational power, was worse by a multiplicative factor of n.

The breakthrough papers by Chen and Sayed (2015); Pu and Garcia (2017); Morral et al. (2017), were the first to address this gap. These papers studied the model where gradients are corrupted by noise, which we also consider in this paper. Chen and Sayed (2015) examined the mean-squared stability and convergence of distributed strategies with fixed step-size over graphs and showed the same performance level as that of a centralized strategy, in the small step-size regime. In Pu and Garcia (2017) it was shown that, for a certain stochastic differential equation paralleling network gradient descent, the performance of centralized and distributed methods were comparable. In Morral et al. (2017), it was proved, for the first time, that distributed gradient descent with an appropriately chosen step-size, asymptotically performs similarly to a centralized method that takes steps in the direction of the sum of the noisy gradients, assuming iterates will remain bounded almost surely. This was the first analysis of a decentralized method for computing the *optimal* solution with performance bounds matching its centralized counterpart.

Both Pu and Garcia (2017) and Morral et al. (2017) were over fixed, undirected graphs with no message loss or delays or asynchronicity. As shown in the paper by Morral et al. (2012), this turns out to be a natural consequence of the analysis of those methods. Indeed, on a technical level, the advantage of working over undirected graphs is that they allow for easy distributed multiplication by doubly-stochastic matrices; it was shown in Morral et al. (2012) that if this property holds only in expectation – that is, if the network nodes can multiply by random stochastic matrices that are only doubly stochastic in expectation – distributed gradient descent will not perform comparably to its centralized counterpart.

In parallel to this work, and in order to reduce communication bottlenecks, Koloskova et al. (2019) propose a decentralized SGD with communication compression that can achieve the centralized baseline convergence rate, up to a constant factor. When the objective functions are smooth but not necessarily convex, Lian et al. (2017) show that Decentralized Parallel Stochastic Gradient Descent (D-PSGD) can asymptotically perform comparably to Centralized PSGD in total computational complexity. However, they argue that D-PSGD requires much less communication cost on the busiest node and hence, can outperform C-PSGD in certain communication regimes. Again, both Koloskova et al. (2019) and Lian et al. (2017) are over fixed undirected graphs, without delays, link failures or asynchronicity. The follow-up work by Lian et al. (2018), extends the D-PSGD to the asynchronous case.

1.2. Our Contribution

We propose an algorithm which we call *Robust Asynchronous Stochastic Gradient Push (RASGP)* for distributed optimization from noisy gradient samples *over directed graphs with message losses, delays, and asynchronous updates.* We will assume gradients are corrupted with additive noise represented by independent random variables, with bounded support, and with finite variance at node i denoted by σ_i^2. Our main result is that the RASGP performs as well as the best bounds on centralized gradient descent that moves in the direction of the sum of noisy gradients of $f_1(\mathbf{z}), \ldots, f_n(\mathbf{z})$. Our results also hold if the underlying graphs are time-varying as long as there are no message losses. We give a brief technical overview of this result next.

We will assume that each function $f_i(\mathbf{z})$ is μ_i-strongly convex with L_i-Lipschitz gradient, where $\sum_i \mu_i > 0$ and $L_i > 0$, $i = 1, \ldots, n$. The RASGP will have every node maintain an estimate of the optimal solution which will be updated from iteration to iteration; we will use $\mathbf{z}_i(k)$ to denote the value of this estimate held by node i at iteration k. We will show that, for each node $i = 1, \ldots, n$,

$$\mathbb{E}\left[\|\mathbf{z}_i(k) - \mathbf{z}^*\|_2^2\right] = \frac{\Gamma_u \sum_{i=1}^n \sigma_i^2}{k(\sum_{i=1}^n \mu_i)^2} + \mathcal{O}_k\left(\frac{1}{k^{1.5}}\right), \tag{2}$$

where $\mathbf{z}^* := \arg\min F(\mathbf{z})$ and Γ_u is the *degree of asynchronicity*, defined as the maximum number of iterations between two consecutive updates of any agent. The leading term matches the best bounds for (centralized) gradient descent that takes steps in the direction of the sum of the noisy gradients of $f_1(\mathbf{z}), \ldots, f_n(\mathbf{z})$, every k/Γ_u iterations (see Nemirovski et al., 2009; Rakhlin et al., 2012). Asymptotically, the performance of the RASGP is network independent: indeed, the only effect of the network or the number of nodes is on the constant factor within the $\mathcal{O}_k\left(1/k^{1.5}\right)$ term above. The asymptotic scaling as $\mathcal{O}_k(1/k)$ is optimal in this setting (Rakhlin et al., 2012).

Consider the case when all the functions are identical, i.e., $f_1(\mathbf{z}) = \cdots = f_n(\mathbf{z})$, and $\Gamma_u = 1$. In this case, letting $\mu = \mu_i$ and $\sigma = \sigma_i$, we have that for each $i = 1, \ldots, n$, (2) reduces to

$$\mathbb{E}\left[\|\mathbf{z}_i(k) - \mathbf{z}^*\|_2^2\right] = \frac{\sigma^2/n}{k\mu^2} + \mathcal{O}_k\left(\frac{1}{k^{1.5}}\right).$$

In other words, asymptotically we get the variance reduction of a centralized method that simply averages the n noisy gradients at each step.

The implication of this result is that one can get the benefit of having n independent processors computing noisy gradients in spite of all the usual problems associated with communications over a network (i.e., message losses, latency, asynchronous updates, one-way communication). Of course, the caveat is that one must wait sufficiently long for the asymptotic decay to "kick in," i.e., for the second term on the right-hand side of (2) to become negligible compared to the first. We leave the analysis of the size of this transient period to future work and note here that it *will* depend on the network and the number of nodes.[1]

The RASGP is a variation on the usual distributed gradient descent where nodes mix consensus steps with steps in the direction of their own gradient, combined with a new step-size trick to deal with asynchrony. It is presented as Algorithm 3 in Section 3. For a formal statement of the results presented above, we refer the reader to Theorem 15 in the body of the paper.

We briefly mention two caveats. The first is that implementation of the RASGP requires each node to use the quantity $\sum_{i=1}^n \mu_i/n$ in setting its local stepsize. This is not a problem in the setting when all functions are the same, but, otherwise, $\sum_{i=1}^n \mu_i/n$ is a global quantity not immediately available to each node. Assuming that node i knows μ_i, one possibility is to use average consensus to compute this quantity in a distributed manner before running the RASGP (for example using the algorithm described in Section 2 of this paper). The second caveat is that, like all algorithms based on the push-sum method, the RASGP requires each node to know its out-degree in the communication graph.

1.3. Organization of This Paper

We conclude this Introduction with Section 1.4, which describes the basic notation we will use throughout the remainder of the paper. Section 2 does not deal directly with the distributed optimization problem we have discussed, but rather introduces the problem of computing the average in the fairly harsh network setting we will consider in this paper. This is an intermediate problem we need to analyze on the way to our main result. Section 3 provides the RASGP algorithm for distributed optimization, and then states and proves our main result, namely the asymptotically network-independent and optimal convergence rate. Results from numerical simulations of our algorithm to illustrate its performance are provided in Section 4, followed by conclusions in Section 5.

1.4. Notations and Definitions

We assume there are n agents $\mathcal{V} = \{1, \ldots, n\}$, communicating through a fixed directed graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, where $\mathcal{E}$ is the set of directed arcs. We assume $\mathcal{G}$ does not have self-loops and is strongly connected.

For a matrix $\mathbf{A}$, we will use A_{ij} to denote its (i,j)th entry. Similarly, v_i and $[v]_i$ will denote the ith entry of a vector $\mathbf{v}$. A matrix is called *stochastic* if it is non-negative and the sum of the elements of each row equals to one. A matrix is *column stochastic* if its transpose is stochastic. To a non-negative matrix $\mathbf{A} \in \mathbb{R}^{n\times n}$ we associate a directed graph $\mathcal{G}_\mathbf{A}$ with vertex set $\mathcal{V}_\mathbf{A} = \{1, 2, \ldots, n\}$ and edge set $\mathcal{E}_\mathbf{A} = \{(i,j) | A_{ji} > 0\}$. In general, such a graph

[1] It goes without saying that no analysis of distributed optimization can be wholly independent of the network or the number of nodes. Indeed, in a network of n nodes, the diameter can be as large as $n-1$, which means that, in the worst case, no bounds on global performance can be obtained during the first $n-1$ steps of any algorithm.

might contain self-loops. Intuitively, this graph corresponds to the information flow in the update $\mathbf{x}(k+1) = \mathbf{A}\mathbf{x}(k)$; indeed, $(i, j) \in \mathcal{E}_{\mathbf{A}}$ if the jth coordinate of $\mathbf{x}(k+1)$ depends on the ith coordinate of $\mathbf{x}(k)$ in this update.

Given a sequence of matrices $\mathbf{A}(0), \mathbf{A}(1), \mathbf{A}(2), \ldots$, we denote by $\mathbf{A}^{k_2:k_1}, k_2 \geq k_1$, the product of elements k_1 to k_2 of the sequence, inclusive, in the following order:

$$\mathbf{A}^{k_2:k_1} = \mathbf{A}(k_2)\mathbf{A}(k_2 - 1) \cdots \mathbf{A}(k_1).$$

Moreover, $\mathbf{A}^{k:k} = \mathbf{A}(k)$.

Node i is an *in-neighbor* of node j, if there is a directed link from i to j. Hence, j would be an *out-neighbor* of node i. We denote the set of in-neighbors and out-neighbors of node i by N_i^- and N_i^+, respectively. Moreover, we denote the number of in-neighbors and out-neighbors of node i with d_i^- and d_i^+, as its *in-degree* and *out-degree*, respectively.

By $x_{\min}$ and $x_{\max}$ we denote $\min_i x_i$ and $\max_i x_i$ respectively, over all possible indices unless mentioned otherwise. We denote a $n \times 1$ column vector of all ones or zeros by $\mathbf{1}_n$ and $\mathbf{0}_n$, respectively. We will remove the subscript when the size is clear from the context.

Let $\mathbf{v} \in \mathbb{R}^d$ be a vector. We denote by $\mathbf{v}^- \in \mathbb{R}^d$ a vector of the same length such that

$$v_i^- = \begin{cases} 1/v_i, & \text{if } v_i \neq 0, \\ 0, & \text{if } v_i = 0. \end{cases}$$

For all the algorithms we describe, we sometimes use the notion of *mass* to denote the value an agent holds, sends or receives. With that in mind, we can think of a value being sent from one node, as a mass being transferred.

We use $\|.\|_p$ to denote the l_p-norm of a vector. We sometimes drop the subscript when referring to the Euclidean l_2 norm.

2. Push-Sum with Delays and Link Failures

In this section we introduce the Robust Asynchronous Push-Sum algorithm (RAPS) for distributed average computation and prove its exponential convergence. Convergence results proved for this algorithm will be used later when we turn to distributed optimization. The algorithm relies heavily on ideas from Hadjicostis et al. (2016) to deal with message losses, delays, and asynchrony. The conference version of this paper Olshevsky et al. (2018) developed RAPS for the delay-free case, and this section may be viewed as an extension of that work.

Pseudocode for the algorithm is given in the box for Algorithm 1. We begin by outlining the operation of the algorithm. Our goal in this section is to compute the average of vectors, one held by each node in the network, in a distributed manner. However, since the RAPS algorithm acts separately in each component, we may, without loss of generality, assume that we want to average scalars rather than vectors. The scalar held by node i will be denoted by $x_i(0)$.

Without loss of generality, we define an iteration by descretizing time into time slots indexed by $k = 0, 1, 2, \ldots$. We assume that during each time slot every agent makes at most one update and processes messages sent in previous time slots.

In the setting of no message losses, no delays, no asynchrony, and a fixed, regular, undirected communication graph, the RAPS can be shown to be equivalent to the much simpler iteration

$$\mathbf{x}(t+1) = \mathbf{W}\mathbf{x}(t),$$

where $\mathbf{W}$ is an irreducible, doubly stochastic matrix with positive diagonal; standard Markov chain theory implies that $x_i(t) \to (1/n)\sum_{i=1}^n x_i(t)$ in this setting. RAPS does essentially the same linear update, but with a considerable amount of modifications. In particular, we use the central idea of the classic push-sum method (Kempe et al., 2003) to deal with directed communication, which suggests to have a separate update equation for the y-variables, which informs us how we should rescale the x-variables; as well as the central idea of Hadjicostis et al. (2018), which is to repeatedly broadcast sums of previous messages to provide robustness against message loss. While the algorithm in Hadjicostis et al. (2018) handles message losses in a synchronous setting, RAPS can handle delays as well as asynchronicity.

Before getting into details, let us provide a simple intuition behind the RAPS algorithm. Each agent i holds a value (mass) x_i and y_i. At the beginning of every iteration, i wants to split its mass between itself and its out-neighbors $j \in N_i^+$. However, to handle message losses, it sends the accumulated x and y mass (running sums which we denote by ϕ_i^x and ϕ_i^y), that i wants to transfer to each of its neighbors, from the start of the algorithm. Therefore, when a neighbor j receives a new accumulated mass from i, it stores it at ρ_{ji}^* and by subtracting the previous accumulated mass ρ_{ji} it had received from i, j obtains all the mass that i has been trying to send since its last successful communication. Then, j updates its x and y mass by adding the new received masses, and finally, updates its estimate of the average to x/y. To handle delays and asynchronicity, timestamps κ_i are attached to messages outgoing from i.

The pseudocode for the algorithm may appear complicated at first glance; this is because of the considerable complexity required to deal with directed communications, message losses, delays, and asynchrony.

We next describe the algorithm in more detail. First, in the course of executing the algorithm, every agent i maintains scalar variables x_i, y_i, z_i, ϕ_i^x, ϕ_i^y, κ_i, ρ_{ij}^x, ρ_{ij}^y and κ_{ij} for $(j,i) \in \mathcal{E}$. The variables x_i and y_i have the same evolution, however y_i is initialized as 1. Therefore, to save space in describing and analyzing the algorithm, we will use the symbol θ, when a statement holds for both x and y. Similarly, when a statement is the same for both variables x and y, we will remove the superscripts x or y. For example, the initialization $\rho_{ji}(0) = 0$ in the beginning of the algorithm means both $\rho_{ji}^x(0) = 0$ and $\rho_{ji}^y(0) = 0$.

We briefly mention the intuitive meaning of the various variables. The number z_i represents node i's estimate of the initial average. The counter $\phi_i^\theta(k)$ is the total θ-value sent by i to each of its neighbors from time 0 to $k-1$. Similarly, $\rho_{ij}^\theta(k)$ is the total θ value that i has received from j up to time $k-1$. The integer κ_i is a timestamp that i attaches to its messages, and the number κ_{ij} tracks the latest timestamp i has received from j.

To obtain an intuition for how the algorithm uses the counters $\phi_i^\theta(k)$ and $\rho_{ij}^\theta(k)$, note that, in line 15 of the algorithm, node i effectively figures out the last θ value sent to it by each of its in-neighbors j, by looking at the increment to the ρ_{ij}^θ. This might seem needlessly involved, but, the underlying reason is that this approach introduces robustness to message losses.

Algorithm 1 Robust Asynchronous Push-Sum (RAPS)

```
Initialize the algorithm with $\mathbf{y}(0) = \mathbf{1}$, $\phi_i(0) = 0$, $\forall i \in \{1, \dots, n\}$ and $\rho_{ij}(0) = 0$, $\kappa_{ij}(0) = 0$, $\forall (j,i) \in \mathcal{E}$.
At every iteration $k = 0, 1, 2, \dots$, for every node $i$:
if node $i$ wakes up then
    $\kappa_i \leftarrow k$;
    $\phi_i^x \leftarrow \phi_i^x + \frac{x_i}{d_i^+ + 1}$, $\phi_i^y \leftarrow \phi_i^y + \frac{y_i}{d_i^+ + 1}$;
    $x_i \leftarrow \frac{x_i}{d_i^+ + 1}$, $y_i \leftarrow \frac{y_i}{d_i^+ + 1}$;
    Node $i$ broadcasts $(\phi_i^x, \phi_i^y, \kappa_i)$ to its out-neighbors in $N_i^+$.
    Processing the received messages
    for $(\phi_j^x, \phi_j^y, \kappa_j')$ in the inbox do
        if $\kappa_j' > \kappa_{ij}$ then
            $\rho_{ij}^{*x} \leftarrow \phi_j^x$, $\rho_{ij}^{*y} \leftarrow \phi_j^y$;
            $\kappa_{ij} \leftarrow \kappa_j'$;
        end if
    end for
    $x_i \leftarrow x_i + \sum_{j \in N_i^-} \left( \rho_{ij}^{*x} - \rho_{ij}^x \right)$, $y_i \leftarrow y_i + \sum_{j \in N_i^-} \left( \rho_{ij}^{*y} - \rho_{ij}^y \right)$;
    $\rho_{ij}^x \leftarrow \rho_{ij}^{*x}$, $\rho_{ij}^y \leftarrow \rho_{ij}^{*y}$,
    $z_i \leftarrow \frac{x_i}{y_i}$;
end if
Other variables remain unchanged.
```

We next describe in words what the pseudocode above does. At every iteration k, if agent i wakes up, it performs the following actions. First, it divides its values x_i, y_i into $d_i^+ + 1$ parts and broadcasts these to its out-neighbors; actually, what it broadcasts are the accumulated running sums ϕ_i^x and ϕ_i^y. Following Kempe et al. (2003), this is sometimes called the "push step."

Then, node i moves on to process the messages in its inbox in the following way. If agent i has received a message from node j that is newer than the last one it has received before, it will store that message in ρ_{ij}^* and discard the older messages. Next, i updates its x and y variables by adding the difference of ρ_{ij}^* with the older value ρ_{ij}, for all in-neighbors j. As mentioned above, this difference is equal to the new mass received. Next, ρ_{ij}^* overwrites ρ_{ij} in the penultimate step. The last step of the algorithm sets z_i to be the rescaled version of x_i: $z_i = x_i / y_i$.

In the remainder of this section, we provide an analysis of the RAPS algorithm, ultimately showing that it converges geometrically to the average in the presence of message losses, asynchronous updates, delays, and directed communication. Our first step is to formulate the RAPS algorithm in terms of a linear update (i.e., a matrix multiplication), which we do in the next subsection.

2.1. Linear Formulation

Next we show that, after introducing some new auxiliary variables, Algorithm 1 can be written in terms of a classical push-sum algorithm (Kempe et al., 2003) on an augmented graph. Since the y-variables have the same evolution as the x-variables, here we only analyze the x-variables.

In our analysis, we will associate with each message an *effective delay.* If a message is sent at time k_1 and is ready to be processed at time k_2, then $k_2 - k_1 \geq 1$ is the effective delay experienced by that message. Those messages that are discarded will not have an effective delay associated with them and are considered as lost.

Next, we will state our assumptions on connectivity, asynchronicity, and message loss.

Assumption 1 *Suppose:*

- *(a) Graph $\mathcal{G}$ is strongly connected and does not have self-loops.*
- *(b) The delays on each link are bounded above by some $\Gamma_{\text{del}} \geq 1$.*
- *(c) Every agent wakes up and performs updates at least once every $\Gamma_u \geq 1$ iterations.*
- *(d) Each link fails at most $\Gamma_f \geq 0$ consecutive times.*
- *(e) Messages arrive in the order of time they were sent. In other words, if messages are sent from node i to j at times k_1 and k_2 with (effective) delays d_1 and d_2, respectively, and $k_1 < k_2$, then we have $k_1 + d_1 < k_2 + d_2$.*

One consequence of Assumption 1 is that the effective delays associated with each message that gets through are bounded above by $\Gamma_d := \Gamma_{\text{del}} + \Gamma_u - 1$. Another consequence is that for each $(i, j) \in \mathcal{E}$, j receives a message from i successfully, at least once every Γ_s iterations where

$$\Gamma_s := \Gamma_u(\Gamma_f + 1) + \Gamma_d \geq 2. \tag{3}$$

Part (e) of Assumption 1 can be assumed without loss of generality. Indeed, observe that outdated messages automatically get discarded in Line 10 of our algorithm. For simplicity, it is convenient to think of those messages as lost. Thus, if this assumption fails in practice, the algorithm will perform exactly as if it had actually held in practice due to Line 10. Making this an assumption, rather than a proposition, lets us slightly simplify some of the arguments and avoid some redundancy throughout this paper.

Let us introduce the following indicator variables: $\tau_i(k)$ for $i \in \{1, \ldots, n\}$ which equals to 1 if node i wakes up at time k, and equals 0 otherwise. Similarly, $\tau_{ij}^l(k)$ for $(i, j) \in \mathcal{E}$, $1 \leq l \leq \Gamma_d$, which is 1 if $\tau_i(k) = 1$ **and** the message sent from node i to j at time k will arrive after experiencing an effective delay of l. [2] Note that if node i wakes up at time k but the message it sends to j is lost, then $\tau_{ij}^l(k)$ will be zero for all l.

We can rewrite the RAPS algorithm with the help of these indicator variables. Let us adopt the notation that $x_i(k)$ refers to x_i at the **beginning** of round k of the algorithm (i.e., before node i has a chance to go through the list of steps outlined in the algorithm

[2] Note the difference between indexing in τ_{ij}^l and ρ_{ji}^x, which are both defined for link $(i, j) \in \mathcal{E}$.

box). We will use the same convention with all of the other variables, e.g., $y_i(k), z_i(k)$, etc. If node i does not wake up at round k, then of course $x_i(k+1) = x_i(k)$.

Now observe that we can write

$$\phi_i^x(k+1) - \phi_i^x(k) = \tau_i(k)\frac{x_i(k)}{d_i^+ + 1}. \tag{4}$$

Likewise, we have

$$x_i(k+1) = x_i(k)\left(1 - \tau_i(k) + \frac{\tau_i(k)}{d_i^+ + 1}\right) + \sum_{j \in N_i^-} \left(\rho_{ij}^x(k+1) - \rho_{ij}^x(k)\right), \tag{5}$$

which can be shown by considering each case ($\tau_i(k) = 1$ or 0); note that we have used the fact that, in the event that node i wakes up at time k, the variable $\rho_{ij}^x(k+1)$ equals the variable ρ_{ij}^{*x} during execution of Line 16 of the algorithm at time k.

Finally, we have that $\forall (i,j) \in \mathcal{E}$, the flows ρ_{ji}^x are updated as follows:

$$\rho_{ji}^x(k+1) = \rho_{ji}^x(k) + \sum_{l=1}^{\Gamma_d} \tau_{ij}^l(k-l)\left(\phi_i^x(k+1-l) - \rho_{ji}^x(k)\right), \tag{6}$$

where we make use of the fact that the sum contains only a single nonzero term, since the messages arrive monotonically. To parse the indices in this equation, note that node i actually broadcasts $\phi_i^x(k+1-l)$ in our notation at iteration $k-l$; by our definitions, $\phi_i^x(k-l)$ is the value of ϕ_i^x at the **beginning** of that iteration. To simplify these relations, we introduce the auxiliary variables u_{ij}^x for all $(i,j) \in \mathcal{E}$, defined through the following recurrence relation:

$$u_{ij}^x(k+1) := \left(1 - \sum_{l=1}^{\Gamma_d} \tau_{ij}^l(k)\right)\Big(u_{ij}^x(k) + \phi_i^x(k+1) - \phi_i^x(k)\Big), \tag{7}$$

and initialized as $u_{ij}^x(0) := 0$. Intuitively, the variables u_{ij}^x represent the "excess mass" of x_i that is yet to reach node j. Indeed, this quantity resets to zero whenever a message is sent that arrives at some point in the future, and otherwise is incremented by adding the broadcasted mass that is lost. Note that node i never knows $u_{ij}^x(k)$, since it has no idea which messages are lost, and which are not; nevertheless, for purposes of analysis, nothing prevents us from considering these variables.

Let us also define the related quantity,

$$v_{ij}^x(k) := u_{ij}^x(k) + \phi_i^x(k+1) - \phi_i^x(k), \qquad \text{for } k \geq 0,$$

and $v_{ij}^x(k) := 0$ for $k < 0$. Intuitively, this quantity may be thought of as a forward-looking estimate of the mass that *will arrive* at node j, if the message sent from node i at time k gets through; correspondingly, it includes not only the previous unsent mass, but the extra mass that will be added at the current iteration.

The key variables for the analysis of our method are the variables we will denote by $x_{ij}^l(k)$. Intuitively, every time a message is sent, but gets lost, we imagine that it has instead arrived into a "virtual node" which holds that mass; once the next message gets through, we imagine that the virtual node has forwarded that mass to its intended destination. This idea originates from Hadjicostis et al. (2016). Because of the delays, however, we need to introduce Γ_d virtual nodes for each such event. If a message is sent from i and arrives at j with effective delay l, we will instead imagine it is received by the virtual node b_{ij}^{l}, then sent to b_{ij}^{l-1} at the next time step, and so forth until it reaches b_{ij}^{1}, and is then forwarded to its destination. These virtual nodes are defined formally later.

Putting that intuition aside, we formally define the variables $x_{ij}^{l}(k)$ via the following set of recurrence relations:

$$x_{ij}^{l}(k+1) := \tau_{ij}^{l}(k)\upsilon_{ij}^{x}(k), \qquad l = \Gamma_d, \tag{8}$$

$$x_{ij}^{l}(k+1) := \tau_{ij}^{l}(k)\upsilon_{ij}^{x}(k) + x_{ij}^{l+1}(k), \qquad 1 \le l < \Gamma_d, \tag{9}$$

and $x_{ij}^{l}(k) := 0$ when both $k \le 0$ and $l = 1, \ldots, \Gamma_d$. To parse these equations, imagine what happens when a message is sent from i to j with effective delay of Γ_d at time k. The content of this message becomes the value of $x_{ij}^{\Gamma_d}$ according to (8); and, in each subsequent step, influences $x_{ij}^{\Gamma_d - 1}, x_{ij}^{\Gamma_d - 2}$, and so forth according to (9). Putting (8) and (9) together, we obtain

$$x_{ij}^{l}(k) = \sum_{t=1}^{\Gamma_d - l + 1} \tau_{ij}^{t+l-1}(k-t)\upsilon_{ij}^{x}(k-t), \tag{10}$$

and particularly,

$$x_{ij}^{1}(k) = \sum_{t=1}^{\Gamma_d} \tau_{ij}^{t}(k-t)\upsilon_{ij}^{x}(k-t). \tag{11}$$

Note that, as is common in many of the equations we will write, only a single term in the sums can be nonzero (this is not obvious at this point and is a result of Lemma 1).

Before proceeding to the main result of this section, we state the following lemma, whose proof is immediate.

Lemma 1 *If $\tau_{ij}^{l}(k) = 1$, the following statements are satisfied:*

(a) $\tau_{ij}^{l'}(k) = 0$ for $l' \ne l$.

(b) If $l > 0$, then $\tau_{ij}^{s}(k+t) = 0$ for $t = 1, \ldots, l$ and $s = 0, \ldots, l-t$.

(c) If $l < \Gamma_d$, then $\tau_{ij}^{s}(k-t) = 0$ for $t = 1, \ldots, \Gamma_d - l$ and $s = l+t, \ldots, \Gamma_d$.

Lemma 2 *If $\tau_{ij}^{l}(k) = 1$ then $x_{ij}^{l'}(k) = 0$ for $l' > l$.*

Proof By Lemma 1(c), $\tau_{ij}^{t+l'-1}(k-t) = 0$ for $t \in \{1, \ldots, \Gamma_d - l' + 1\}$. Hence, by (10) we have,

$$x_{ij}^{l'}(k) = \sum_{t=1}^{\Gamma_d - l' + 1} \tau_{ij}^{t+l'-1}(k-t)\upsilon_{ij}^{x}(k-t) = 0.$$

■

The next lemma is essentially a restatement of the observation that the content of every $x_{ij}^{l'}$ eventually "passes through" x_{ij}^1.

Lemma 3 *If $\tau_{ij}^l(k-l)=1$, $l \geq 1$, we have,*

$$\sum_{l'=1}^{l} x_{ij}^{l'}(k-l) = \sum_{t=1}^{l} x_{ij}^1(k-t).$$

Proof We will show $x_{ij}^1(k-t) = x_{ij}^{l-t+1}(k-l)$ for $t=1,\ldots,l$. For $t=l$ the equality is trivial. Now suppose $t<l$. By Lemma 1(a) we have $\tau_{ij}^{l-t}(k-l)=0$. Moreover, by part (b) of the same lemma we have, $\tau_{ij}^{s'}(k-l+t')=0$ for $t'=1,\ldots,l-t-1$ and $s'=l-t-t'$. Hence, $x_{ij}^{l-t-t'+1}(k-l+t') = x_{ij}^{l-t-t'}(k-l+t'+1)$. Combining these equations for $t'=0,\ldots,l-t-1$, we get $x_{ij}^1(k-t) = x_{ij}^{l-t+1}(k-l)$. ■

The following lemma is the key step of a linear formulation of RAPS.

Lemma 4 *For $k=0,1,\ldots$ and $(i,j)\in\mathcal{E}$ we have:*

$$\rho_{ji}^x(k+1) - \rho_{ji}^x(k) = x_{ij}^1(k), \tag{12}$$

$$u_{ij}^x(k+1) + \rho_{ji}^x(k+1) + \sum_{l=1}^{\Gamma_d} x_{ij}^l(k+1) = \phi_i^x(k+1). \tag{13}$$

Parsing these equations, (12) simply states that the value of $x_{ij}^1(k)$ can be thought of as impacting ρ_{ji}^x at time k; recall that the content of $x_{ij}^1(k)$ is a message that was sent from node i to j at time $k-l$ with an effective delay of l, for some $1 \leq l \leq \Gamma_d$ (cf. Equation 11). On the other hand, (13) may be thought of a "conservation of mass" equation. All the mass that has been sent out by node i has either: (i) been lost (in which case it is in u_{ij}^x), (ii) affected node j (in which case it is in ρ_{ji}^x), or (iii) is in the process of reaching node j but delayed (in which case it is in some x_{ij}^l).

Although this lemma is arguably obvious, a formal proof is surprisingly lengthy. For this reason, we relegate it to the Appendix.

We next write down a matrix form of our updates. As a first step, define the $(n+m')\times 1$ column vector $\boldsymbol{\chi}(k) := [\mathbf{x}(k)^\top, \mathbf{x}^1(k)^\top, \ldots, \mathbf{x}^{\Gamma_d}(k)^\top, \mathbf{u}^x(k)^\top]^\top$, where $m' := (\Gamma_d+1)m$, $m := |\mathcal{E}|$, $\mathbf{x}(k)$ collects all $x_i(k)$, $\mathbf{x}^l(k)$ collects all $x_{ij}^l(k)$ and, $\mathbf{u}^x(k)$ collects all $u_{ij}^x(k)$. Define $\boldsymbol{\psi}(k)$ by collecting y-values similarly.

Now, we have all the tools to show the linear evolution of $\boldsymbol{\chi}(k)$. By Equations (4), (5) and (12) we have,

$$x_j(k+1) = x_j(k)\left(1 - \tau_j(k) + \frac{\tau_j(k)}{d_j^+ + 1}\right) + \sum_{i\in N_j^-} x_{ij}^1(k). \tag{14}$$

Moreover, by the definitions of x_{ij}, υ_{ij} and (4) it follows,

$$\begin{aligned} x_{ij}^{\Gamma_d}(k+1) &= \tau_{ij}^{\Gamma_d}(k)\left[u_{ij}^x(k) + \frac{x_i(k)}{d_i^+ + 1}\right], \\ x_{ij}^{l}(k+1) &= \tau_{ij}^{l}(k)\left[u_{ij}^x(k) + \frac{x_i(k)}{d_i^+ + 1}\right] + x_{ij}^{l+1}(k). \end{aligned} \tag{15}$$

Finally, by (4) and (7) we obtain,

$$u_{ij}^x(k+1) = \big(1 - \sum_{l=1}^{\Gamma_d} \tau_{ij}^l(k)\big)\Big(u_{ij}^x(k) + \tau_i(k)\frac{x_i(k)}{d_i^+ + 1}\Big). \tag{16}$$

Using (14) to (16) we can write the evolution of $\boldsymbol{\chi}(k)$ and $\boldsymbol{\psi}(k)$ in the following linear form:

$$\begin{aligned} \boldsymbol{\chi}(k+1) &= \mathbf{M}(k)\boldsymbol{\chi}(k), \\ \boldsymbol{\psi}(k+1) &= \mathbf{M}(k)\boldsymbol{\psi}(k), \end{aligned} \tag{17}$$

where $\mathbf{M}(k) \in \mathbb{R}^{(n+m')\times(n+m')}$ is an appropriately defined matrix.

We have thus completed half of our goal: we have shown how to write RAPS as a linear update. Next, we show that the corresponding matrices are column-stochastic.

Lemma 5 $\mathbf{M}(k)$ *is column stochastic and its positive elements are at least* $1/(\max_i\{d_i^+\}+1)$. *Moreover, for* $i = 1, \ldots, n$, $M_{ii}(k)$ *are positive.*

This lemma can be proved "by inspection." Indeed, $\mathbf{M}(k)$ is column stochastic if and only if, for every $\boldsymbol{\chi}(k)$, we have $\mathbf{1}^T\boldsymbol{\chi}(k+1) = \mathbf{1}^T\boldsymbol{\chi}(k)$. Thus one just needs to demonstrate that no mass is ever "lost," i.e., that a decrease/increase in the value of one node is always accompanied by an increase/decrease of the value of another node, which can be done just by inspecting the equations. A formal proof is nonetheless given next.

Proof To show that $\mathbf{M}(k)$ is column stochastic, we study how each element of $\boldsymbol{\chi}(k)$ influences $\boldsymbol{\chi}(k+1)$.

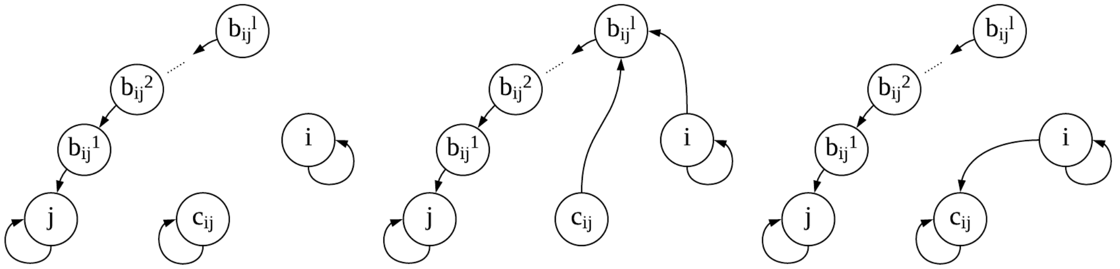
For $i = 1, \ldots, n$, the ith column of $\mathbf{M}(k)$ represents how $x_i(k)$ influences $\boldsymbol{\chi}(k+1)$. We will use (14) to (16) to find these coefficients.

First, $x_i(k)$ influences $x_i(k+1)$ with the coefficient $1 - \tau_i(k) + \tau_i(k)/(d_i^+ + 1) > 0$. For $j \in N_i^+$, $x_i(k)$ influences $x_{ij}^l(k+1)$ by $\tau_{ij}^l(k)/(d_i^+ + 1)$ and $u_{ij}^x(k+1)$ with coefficient $(\tau_i(k) - \sum_{l=1}^{\Gamma_d} \tau_i(k)\tau_{ij}^l(k))/(d_i^+ + 1)$. Summing these coefficients up results in 1.

For $l = 2, \ldots, \Gamma_d$, $(i,j) \in \mathcal{E}$, $x_{ij}^l(k)$ influences $x_{ij}^{l-1}(k+1)$ with coefficient 1 and $x_{ij}^1(k)$ influences $x_j(k+1)$ with coefficient 1.

Finally, $u_{ij}^x(k)$ influences $x_{ij}^l(k+1)$ with coefficient $\tau_{ij}^l(k)$ and $u_{ij}^x(k+1)$ with $(1 - \sum_{l=1}^{d} \tau_{ij}^l(k))$, which sum up to 1.

Note that all the coefficients above are at least $1/(\max_i\{d_i^+\}+1)$. ■



(a) $\tau_i(k) = 0$.

(b) $\tau_{ij}^l(k) = 1$.

(c) $\tau_i(k) = 1$, $\tau_{ij}^l(k) = 0, \forall l$.

Figure 2: Augmented graph $\mathcal{H}(k)$ for different scenarios.

An important result of this lemma is the sum preservation property, i.e.,

$$\begin{aligned} \sum_{i=1}^{n+m'} \chi_i(k) &= \sum_{i=1}^{n} x_i(0), \\ \sum_{i=1}^{n+m'} \psi_i(k) &= n. \end{aligned} \tag{18}$$

For further analysis, we augment the graph $\mathcal{G}$ to $\mathcal{H}(k) := \mathcal{G}_{\mathbf{M}(k)} = (\mathcal{V}_A, \mathcal{E}_A(k))$ by adding the following virtual nodes: b_{ij}^l for $l = 1, \ldots, \Gamma_d$ and $(i, j) \in \mathcal{E}$, which hold the values x_{ij}^l and y_{ij}^l; We also add the nodes c_{ij} for $(i, j) \in \mathcal{E}$ which hold the values u_{ij}^x and u_{ij}^y.

In $\mathcal{H}(k)$, there is a link from b_{ij}^l to b_{ij}^{l-1} for $1 < l \leq d$ and from b_{ij}^1 to j as they forward their values to the next node. Moreover, if $\tau_{ij}^l(k) = 1$ for some $1 \leq l \leq \Gamma_d$, then there is a link from both c_{ij} and i to b_{ij}^l.

If $\tau_{ij}^l(k) = 0$ for $1 \leq l \leq \Gamma_d$ then c_{ij} has a self loop, and if also $\tau_i(k) = 1$, there's a link from i to c_{ij}. All non-virtual agents $i \in \mathcal{V}$, have self-loops all the time (see Fig. 2).

Recursions (17) and Lemma 5 may thus be interpreted as showing that the RAPS algorithm can be thought of as a push-sum algorithm over the augmented graph sequence $\{\mathcal{H}(k)\}$, where each agent (virtual and non-virtual) holds an x-value and a y-value which evolve similarly and in parallel.

2.2. Exponential Convergence

The main result of this section is exponential convergence of RAPS to initial average, stated next.

Theorem 6 *Suppose Assumption 1 holds. Then RAPS converges exponentially to the initial mean of agent values. i.e.,*

$$\left| z_i(k) - \frac{1}{n} \sum_{i=1}^{n} x_i(0) \right| \leq \delta \lambda^k \|\mathbf{x}(0)\|_1,$$

where $\delta := \frac{1}{1 - n\alpha^6}$, $\lambda := (1 - n\alpha^6)^{1/(2n\Gamma_s)}$ *and* $\alpha := (1/n)^{n\Gamma_s}$.

It is worth mentioning that even though $1/(1-\lambda) = \mathcal{O}(n^{p(n)})$ where $p(n) = \mathcal{O}(n)$, this is a bound for a worst case scenario and on average, as it can be seen in numerical simulations, RAPS performs better. Moreover, when the graph $\mathcal{G}$ satisfies certain properties, such as regularity, and also there is no link delays and failures, we have $1/(1-\lambda) = \mathcal{O}(n^3)$ (see Theorem 1 in Nedic and Olshevsky, 2016). More broadly, that paper establishes that $1/(1-\lambda)$ will scale with the mixing rate of the underlying Markov process.

Unfortunately, this theorem does not follow immediately from standard results on exponential convergence of push-sum. The reason is that the connectivity conditions assumed for such theorems are not satisfied here: there will not always be paths leading to virtual nodes from non-virtual nodes. Nevertheless, with some suitable modifications, the existence of paths from virtual nodes to other virtual nodes is sufficient, as we will show next.

Before proving the theorem, we need the following lemmas and definitions. Given a sequence of graphs $\mathcal{G}^0, \mathcal{G}^1, \mathcal{G}^2, \ldots$, we will say node b is reachable from node a in time period k_1 to k_2 ($k_1 < k_2$), if there exists a sequence of directed edges $e_{k_1}, e_{k_1+1}, \ldots, e_{k_2}$ such that e_k is in $\mathcal{G}^k$, the destination of e_k is the origin of e_{k+1} for $k_1 \leq k < k_2$, and the origin of e_{k_1} is a and the destination of e^{k_2} is b.

Our first lemma provides a standard lower bound on the entries of the column-stochastic matrices from (17).

Lemma 7 $\mathbf{M}^{k+n\Gamma_s-1:k}$ *has positive first* n *rows, for any* $k \geq 0$. *The positive elements of this matrix are at least*

$$\alpha = (1/n)^{n\Gamma_s}.$$

Proof By Lemma 5, each node $j \in \mathcal{V}$ has self-loops at every iteration in the augmented graph $\mathcal{H}$. Since $\mathcal{G}$ is strongly connected, the set of reachable non-virtual nodes from any node $a_h \in \mathcal{V}_A$ strictly increases every Γ_s iterations. Hence, $\mathbf{M}^{k+n\Gamma_s-1:k}$ has positive first n rows. Moreover, since all positive elements of M are at least $1/n$, the positive elements of $\mathbf{M}^{k+n\Gamma_s-1:k}$ are at least $(1/n)^{n\Gamma_s}$. ■

Next, we give a reformulation of the push-sum update that will be key to showing the exponential convergence of the algorithm. The proof is a minor variation of Lemma 4 in Nedic and Olshevsky (2016).

Lemma 8 *Consider the vectors* $\mathbf{u}(k) \in \mathbb{R}^d$, $\mathbf{v}(k) \in \mathbb{R}^d_+$ *and square matrix* $\mathbf{A}(k) \in \mathbb{R}^{d\times d}_+$, *for* $k \geq 0$ *such that,*

$$\begin{aligned} \mathbf{u}(k+1) &= \mathbf{A}(k)\mathbf{u}(k), \\ \mathbf{v}(k+1) &= \mathbf{A}(k)\mathbf{v}(k). \end{aligned} \tag{19}$$

Also suppose $u_i(k) = 0$ *if* $v_i(k) = 0$, $\forall k, i$. *Define* $\mathbf{u}^-(k) \in \mathbb{R}^d$ *as:*

$$u_i^-(k) := \begin{cases} 1/u_i(k), & \text{if } u_i(k) \neq 0, \\ 0, & \text{if } u_i(k) = 0. \end{cases}$$

Define $\mathbf{r}(k) := \mathbf{u}(k) \circ \mathbf{v}^-(k)$, *where* $\circ$ *denotes the element-wise product of two vectors. Then we have,*

$$\mathbf{r}(k+1) = \mathbf{B}(k)\mathbf{r}(k),$$

where $\mathbf{B}(k) \in \mathbb{R}_+^{d\times d}$ *is defined as,*

$$\mathbf{B}(k) := \mathrm{diag}(\mathbf{v}^-(k+1))\mathbf{A}(k)\mathrm{diag}(\mathbf{v}(k)).$$

Proof Since $u_i(k) = 0$ if $v_i(k) = 0$, $u_i(k) = r_i(k)v_i(k)$ holds for all i, k. Substituting in (19) we obtain,

$$r_i(k+1)v_i(k+1) = \sum_{j=1}^{d} A_{ij}(k)r_j(k)v_j(k).$$

Since, by definition $r_i(k) = 0$ if $v_i(k) = 0$, $\forall k, i$, we get

$$r_i(k+1) = v_i^-(k+1)\sum_{j=1}^{d} A_{ij}(k)r_j(k)v_j(k).$$

Therefore,

$$\mathbf{r}(k+1) = \mathrm{diag}(\mathbf{v}^-(k+1))\mathbf{A}(k)\mathrm{diag}(\mathbf{v}(k))\mathbf{r}(k).$$

■

Our next corollary, which follows immediately from the previous lemma, characterizes the dichotomy inherent in push-sum with virtual nodes: every row either adds up to one or zero.

Corollary 9 *Consider the matrix* $\mathbf{B}(k)$ *defined in Lemma 8. Let us define the index set* $J^k := \{i|\, v_i(k) \neq 0\}$. *If* $i \notin J^k$, *the ith column of* $\mathbf{B}(k)$ *and ith row of* $\mathbf{B}(k-1)$ *only contain zero entries. Moreover,*

$$\begin{aligned}\mathbf{B}(k)\mathbf{1}_d &= \mathrm{diag}(\mathbf{v}^-(k+1))\mathbf{A}(k)\mathbf{v}(k)\\ &= \mathrm{diag}(\mathbf{v}^-(k+1))\mathbf{v}(k+1) = \begin{bmatrix} 1 \text{ or } 0\\ \vdots \\ 1 \text{ or } 0\end{bmatrix}.\end{aligned}$$

Hence, the ith row of $\mathbf{B}(k)$ *sums to* 1 *if and only if* $\mathbf{v}_i(k+1) \neq 0$ *or* $i \in J^{k+1}$.

Our next lemma characterizes the relationship between zero entries in the vectors $\boldsymbol{\chi}(k)$ and $\boldsymbol{\psi}(k)$.

Lemma 10 $\chi_h(k) = 0$ *whenever* $\psi_h(k) = 0$ *for* $h = 1, \ldots, n+m'$, $k \geq 0$.

Proof First we note that $\boldsymbol{\psi}(0) = [\mathbf{1}_n^\top, \mathbf{0}_{m'}^\top]^\top$ and each node $i \in \mathcal{V}$ has a self-loop in graph $\mathcal{H}(k)$ for all $k \geq 0$; hence, $\psi_h(k) \geq 0$ for all h and particularly, $\psi_i(k) > 0$ for $i = 1, \ldots, n$. Now suppose $h > n$ and corresponds to a virtual agent $a_h \in \mathcal{V}_A$. If $\psi_h(k) = 0$, it means a_h has already sent all its y-value to another node or has not received any y-value yet. In either case, that node also has no remaining x-value as well and $\chi_h(k) = 0$. ■

Let us define $\boldsymbol{\psi}^-(k) \in \mathbb{R}^{n+m'}$, $k \geq 0$ by

$$\psi_i^-(k) := \begin{cases} 1/\psi_i(k), & \text{if } \psi_i(k) \neq 0, \\ 0, & \text{if } \psi_i(k) = 0. \end{cases} \tag{20}$$

Moreover, we define the vector $\mathbf{z}(k)$ by setting $\mathbf{z}(k) := \boldsymbol{\chi}(k) \circ \boldsymbol{\psi}^-(k)$. By (17) and Lemma 10, we can use Lemma 8 to obtain,

$$\mathbf{z}(k+1) = \mathbf{P}(k)\mathbf{z}(k),$$

where $\mathbf{P}(k) := \text{diag}(\boldsymbol{\psi}^-(k+1))\mathbf{M}(k)\text{diag}(\boldsymbol{\psi}(k))$. Let us define

$$I^k := \{i | \, \psi_i(k) > 0\}.$$

Then, by Corollary 9 we have each $z_i(k+1)$, $i \in I^{k+1}$, is a convex combination of $z_j(k)$'s for $j \in I^k$. Therefore,

$$\begin{aligned} \max_{i \in I^{k+1}} z_i(k+1) &\leq \max_{i \in I^k} z_i(k), \\ \min_{i \in I^{k+1}} z_i(k+1) &\geq \min_{i \in I^k} z_i(k). \end{aligned} \tag{21}$$

These equations will be key to the analysis of the algorithm. We stress that we have not shown that the quantity $\min_i z_i(k)$ is non-decreasing; rather, we have shown that the related quantity, where the minimum is taken over I^k, the set of nonzero entries of $\boldsymbol{\psi}(k)$, is non-increasing.

Our next lemma provides lower and upper bounds on the entries of the vector $\boldsymbol{\psi}(k)$.

Lemma 11 *For $k \geq 0$ and $1 \leq i \leq n$ we have:*

$$n\alpha \leq \psi_i(k) \leq n.$$

Moreover, for $n+1 \leq h \leq n+m'$ and $k \geq 1$ we have either $\psi_h(k) = 0$ or,

$$n\alpha^2 \leq \psi_h(k) \leq n.$$

Proof We have,

$$\boldsymbol{\psi}(k) = \mathbf{M}^{k-1:0} \begin{bmatrix} \mathbf{1}_n \\ \mathbf{0}_{m'} \end{bmatrix},$$

If $k < n\Gamma_s$, positive entries of $\mathbf{M}^{k-1:0}$ are at least $(1/n)^k$. Hence, positive entries of $\boldsymbol{\psi}(k)$ are at least,

$$\left(\frac{1}{n}\right)^k \geq \left(\frac{1}{n}\right)^{n\Gamma_s - 1} = n\alpha.$$

Now suppose $k \geq n\Gamma_s$. $\mathbf{M}^{k-1:0}$ is the product of $\mathbf{M}^{k-1:k-n\Gamma_s}$ and another column stochastic matrix. By Lemma 7, $\mathbf{M}^{k-1:k-n\Gamma_s}$ has positive first n rows, and positive entries of at least

α. Thus, $\mathbf{M}^{k-1:0}$ has positive first n rows, and positive entries of at least α as well. We obtain for $1 \leq i \leq n$,

$$\psi_i(k) \geq n\alpha, \text{ for } k \geq 1.$$

For $n+1 \leq h \leq n+m'$, suppose ψ_h corresponds to a virtual node a_h corresponding to some link $(i,j) \in \mathcal{E}$. If $\psi_h(k)$ is positive, it is carrying a value sent from i at $k - n\Gamma_s$ or later, which has experienced link failure or delays. This is because each value gets to its destination after at most Γ_s iterations. Since i has self-loops all the time, a_h is reachable from i in period $k - n\Gamma_s$ to $k-1$; Hence, $M_{hi}^{k-1:k-n\Gamma_s} \geq \alpha$, and it follows,

$$\psi_h(k) \geq \alpha\psi_i(k - n\Gamma_s) \geq n\alpha^2.$$

Also, due to sum preservation property, we have $\psi_h(k) \leq n$, for all h and $k \geq 0$. ■

Using Lemma 8 again, it follows,

$$\mathbf{z}(k + n\Gamma_s) = \hat{\mathbf{P}}(k)\mathbf{z}(k),$$

where,

$$\hat{\mathbf{P}}(k) := \text{diag}(\boldsymbol{\psi}^-(k + n\Gamma_s))\mathbf{M}^{k+nLs-1:k}\text{diag}(\boldsymbol{\psi}(k)). \tag{22}$$

Next, we are able to find a lower bound on the positive elements of $\hat{\mathbf{P}}(k)$. The proof of the following corollary is immediate.

Corollary 12 *By* (22) *and Lemma 11 we have:*

(a) $\hat{P}_{ij}(k) > 0$ *for* $1 \leq i,j \leq n$.

(b) Positive entries of first n columns of $\hat{P}(k)$ are at least $(1/n)\alpha(n\alpha) = \alpha^2$. Similarly, the last m' columns have positive entries of at least α^3.

(c) For $h > n$, if $h \in I^{k+n\Gamma_s}$ then $\hat{P}_{hi}(k) > 0$ for some $1 \leq i \leq n$.

Our next lemma, which is the final result we need before proving the exponential convergence rate of RAPS, provides a quantitative bound for how multiplication by the matrix $\mathbf{P}$ shrinks the range of a vector.

Lemma 13 *Let $t \geq 0$ and $\{\mathbf{u}(k)\}_{k\geq 0} \in \mathbb{R}^{n+m'}$ be a sequence of vectors such that,*

$$\mathbf{u}(k+1) = \hat{\mathbf{P}}(kn\Gamma_s + t)\mathbf{u}(k).$$

Define

$$s_t(k) := \max_{i \in I^{kn\Gamma_s+t}} u_i(k) - \min_{i \in I^{kn\Gamma_s+t}} u_i(k).$$

Then,

$$s_t(k+2) \leq (1 - n\alpha^6)s_t(k).$$

Proof Let us define

$$r_t(k) := \max_{1\le i\le n} u_i(k) - \min_{1\le i\le n} u_i(k).$$

By Corollary 12 for $j \in I^{(k+1)n\Gamma_s+t}$, the jth row of $\hat{\mathbf{P}}(kn\Gamma_s + t)$ has at least one positive entry in the first n columns. Thus, because $u_j(k+1)$ is maximized/minimized when all of the weight is put on the largest/smallest possible entry of $u_j(k)$, we have:

$$\begin{aligned} u_j(k+1) &\le \alpha^3 \max_{1\le i\le n} u_i(k) + (1-\alpha^3) \max_{i\in I^{kn\Gamma_s+t}} u_i(k), \\ u_j(k+1) &\ge \alpha^3 \min_{1\le i\le n} u_i(k) + (1-\alpha^3) \min_{i\in I^{kn\Gamma_s+t}} u_i(k), \end{aligned}$$

Therefore,

$$s_t(k+1) \le \alpha^3 r_t(k) + (1-\alpha^3) s_t(k). \tag{23}$$

Moreover, by a similar argument for $j \le n$,

$$\begin{aligned} u_j(k+1) &\le \alpha^3 \sum_{i=1}^{n} u_i(k) + (1-n\alpha^3) \max_{i\in I^{kn\Gamma_s+t}} u_i(k), \\ u_j(k+1) &\ge \alpha^3 \sum_{i=1}^{n} u_i(k) + (1-n\alpha^3) \min_{i\in I^{kn\Gamma_s+t}} u_i(k). \end{aligned}$$

Thus,

$$r_t(k+1) \le (1-n\alpha^3) s_t(k).$$

Combining with (23) and noting that $r_t(k) \le s_t(k)$ and $s_t(k+1) \le s_t(k)$ we obtain,

$$\begin{aligned} s_t(k+2) &\le \alpha^3(1-n\alpha^3)s_t(k) + (1-\alpha^3)s_t(k+1) \\ &\le \alpha^3(1-n\alpha^3)s_t(k) + (1-\alpha^3)s_t(k) \\ &= (1-n\alpha^6)s_t(k). \end{aligned}$$

■

Proof of Theorem 6 Using Lemma 13 with $t = 0$ and $\mathbf{u}(k) = \mathbf{z}(kn\Gamma_s)$ we get $s_0(k) \le (1-n\alpha^6)^{\lfloor k/2 \rfloor} s_0(0)$ and $\lim_{k\to\infty} s_0(k) = 0$. Moreover by (21), $\mathbf{z}_{\max}(k)$ is a non-increasing sequence and by $\mathbf{z}_{\min}(k)$, is non-decreasing. Thus,

$$\lim_{k\to\infty,\, h\in I^k} \mathbf{z}_h(k) = L_\infty. \tag{24}$$

We have:

$$\begin{aligned} L_\infty &= L_\infty \lim_{k\to\infty} \frac{\sum_{i=1}^{n+m'} \psi_i(k)}{\sum_{i=1}^{n+m'} \psi_i(k)} \\ &= \lim_{k\to\infty} \left(\frac{\sum_{i=1}^{n+m'} z_i(k)\psi_i(k)}{n} + \frac{\sum_{i=1}^{n+m'} (L_\infty - z_i(k))\psi_i(k)}{n} \right) \\ &= \lim_{k\to\infty} \left(\frac{\sum_{i=1}^{n+m'} \chi_i(k)}{n} + \frac{\sum_{i=1}^{n+m'} (L_\infty - z_i(k))\psi_i(k)}{n} \right) \\ &= \frac{\sum_{i=1}^{n} x_i(0)}{n}. \end{aligned}$$

In the above, we used (18) and (24), the boundedness of $\psi_i(k)$, and the fact that $\psi_i(k) = 0$ for $i \notin I^k$.

Finally, to show the exponential convergence rate, we go back to $s_0(k)$. We have for $k \geq 1$,

$$\begin{aligned}
s_0(k) &\leq (1 - n\alpha^6)^{\lfloor k/2 \rfloor} s_0(0) \leq (1 - n\alpha^6)^{(k-1)/2} s_0(0),\\
s_0(0) &\leq \sum_{i=1}^{n+m'} |z_i(0)| = \sum_{i=1}^{n} |x_i(0)| = \|\mathbf{x}(0)\|_1,
\end{aligned}$$

where the first equality holds because $I^0 = \{1, \ldots, n\}$ and $y_i(0) = 1$. Therefore, we have for $i \in I^k$,

$$\begin{aligned}
\left| z_i(k) - \frac{\mathbf{1}^\top \mathbf{x}(0)}{n} \right| &\leq z_{\max}(k) - z_{\min}(k)\\
&\leq s_0(\lfloor k/n\Gamma_s \rfloor)\\
&\leq (1 - n\alpha^6)^{(\lfloor \frac{k}{n\Gamma_s} \rfloor - 1)/2} \|\mathbf{x}(0)\|_1\\
&\leq (1 - n\alpha^6)^{(\frac{k}{n\Gamma_s} - 1 - 1)/2} \|\mathbf{x}(0)\|_1\\
&= \frac{1}{1 - n\alpha^6} \left((1 - n\alpha^6)^{1/(2n\Gamma_s)} \right)^k \|\mathbf{x}(0)\|_1\\
&= \delta \lambda^k \|\mathbf{x}(0)\|_1.
\end{aligned}$$

where $\delta = \frac{1}{1 - n\alpha^6}$ and $\lambda = (1 - n\alpha^6)^{1/(2n\Gamma_s)}$. Note that $\{1, \ldots, n\} \subseteq I^k$, $\forall k$. ■

Remark: Observe that our proof did not really use the initialization $\boldsymbol{\psi}(0) = \mathbf{1}$, except to observe that the elements $\boldsymbol{\psi}(0)$ are positive, add up to n, and the implication that $\boldsymbol{\psi}(k)$ satisfies the bounds of Lemma 11. In particular, the same result would hold if we viewed time 1 as the initial point of the algorithm (so that $\boldsymbol{\psi}(1)$ is the initialization), or similarly any time k. We will use this observation in the next subsection.

2.3. Perturbed Push-Sum

In this subsection, we begin by introducing the Perturbed Robust Asynchronous Push-Sum algorithm, obtained by adding a perturbation to the x-values of (non-virtual) agents at the beginning of every iteration they wake up.

We show that, if the perturbations are bounded, the resulting $\mathbf{z}(k)$ nevertheless tracks the average of $\boldsymbol{\chi}(k)$ pretty well. Such a result is a key step towards analyzing distributed optimization protocols. In this general approach to the analyses of distributed optimization methods, we follow Ram et al. (2010) where it was first adopted; see also Nedic and Olshevsky (2016) and Nedic and Olshevsky (2015) where it was used.

Adopting the notations introduced earlier and by the linear formulation (17) we have,

$$\boldsymbol{\chi}(k+1) = \mathbf{M}(k)(\boldsymbol{\chi}(k) + \boldsymbol{\Delta}(k)), \qquad \text{for } k \geq 0,$$

Algorithm 2 Perturbed Robust Asynchronous Push-Sum

```
1: Initialize the algorithm with $\mathbf{y}(0) = \mathbf{1}$, $\phi_i(0) = 0$, $\forall i \in \{1, \ldots, n\}$ and $\rho_{ij}(0) = 0$,
   $\kappa_{ij}(0) = 0$, $\forall (j, i) \in \mathcal{E}$ and $\boldsymbol{\Delta}(0) = \mathbf{0}$.
2: At every iteration $k = 0, 1, 2, \ldots$, for every node $i$:
3: if node $i$ wakes up then
4:     $x_i \leftarrow x_i + \Delta_i(k)$;
5:     Lines 4 to 17 of Algorithm 1
6: end if
7: Other variables remain unchanged.
```

where $\boldsymbol{\Delta}(k) \in \mathbb{R}^{n+m'}$ collects all perturbations $\Delta_i(k)$ in a column vector with $\Delta_h(k) := 0$ for $n < h \leq n + m'$. We may write this in a convenient form as follows.

$$\begin{aligned}\boldsymbol{\chi}(k+1) &= \mathbf{M}(k)(\boldsymbol{\chi}(k) + \boldsymbol{\Delta}(k)) \\ &= \sum_{t=1}^{k} \mathbf{M}^{k:t} \boldsymbol{\Delta}(t) + \mathbf{M}^{k:0} \boldsymbol{\chi}(0).\end{aligned}$$

Define for $k \geq 1$,

$$\begin{aligned}\boldsymbol{\chi}^t(k) &:= \mathbf{M}^{k-1:t} \boldsymbol{\Delta}(t), \qquad 1 \leq t \leq k, \\ \boldsymbol{\chi}^0(k) &:= \mathbf{M}^{k-1:0} \boldsymbol{\chi}(0), \qquad t = 0.\end{aligned} \tag{25}$$

We obtain,

$$\boldsymbol{\chi}(k) = \sum_{t=0}^{k-1} \boldsymbol{\chi}^t(k), \qquad k \geq 1. \tag{26}$$

Define $\mathbf{z}^t(k) := \boldsymbol{\chi}^t(k) \circ \boldsymbol{\psi}^-(k)$ for $0 \leq t \leq k$ (cf. Equation 20). We have

$$\mathbf{z}(k) = \sum_{t=0}^{k-1} \mathbf{z}^t(k). \tag{27}$$

We may view each $\mathbf{z}^t(k)$ as the outcome of a push-sum algorithm, initialized at time t, and apply Theorem 6. This immediately yields the following result, with part (b) an immediate consequence of part (a).

Theorem 14 *Suppose Assumption 1 holds. Consider the sequence $\{z_i(k)\}$, $1 \leq i \leq n$, generated by Algorithm 2. Then,*

(a) For $k = 1, 2, \ldots$

$$\left| z_i(k) - \frac{\mathbf{1}^\top \boldsymbol{\chi}(k)}{n} \right| \leq \delta \lambda^k \|\mathbf{x}(0)\|_1 + \sum_{t=1}^{k-1} \delta \lambda^{k-t} \|\boldsymbol{\Delta}(t)\|_1.$$

(b) If $\lim_{t \to \infty} \|\boldsymbol{\Delta}(t)\|_1 = 0$ then,

$$\lim_{k \to \infty} \left| z_i(k) - \frac{\mathbf{1}^\top \boldsymbol{\chi}(k)}{n} \right| = 0.$$

3. Robust Asynchronous Stochastic Gradient-Push (RASGP)

In this section we present the main contribution of this paper, a distributed stochastic gradient method with asymptotically network-independent and optimal performance over directed graphs which is robust to asynchrony, delays, and link failures.

Recall that we are considering a network $\mathcal{G}$ of n agents whose goal is to cooperatively solve the following minimization problem

$$\text{minimize } F(\mathbf{z}) := \sum_{i=1}^{n} f_i(\mathbf{z}), \qquad \text{over } \mathbf{z} \in \mathbb{R}^d,$$

where each $f_i : \mathbb{R}^d \to \mathbb{R}$ is a strongly convex function only known to agent i. We assume agent i has the ability to obtain noisy gradients of the function f_i.

The RASGP algorithm is given as Algorithm 3. Note that we use the notation $\hat{\mathbf{g}}_i(k)$ for a noisy gradient of the function $f_i(\mathbf{z})$ at $\mathbf{z}_i(k)$ i.e.,

$$\hat{\mathbf{g}}_i(k) = \mathbf{g}_i(k) + \boldsymbol{\varepsilon}_i,$$

where $\mathbf{g}_i(k) := \nabla f_i(\mathbf{z}_i(k))$ and $\boldsymbol{\varepsilon}_i$ is a random vector.

The RASGP is based on a standard idea of mixing consensus and gradient steps, first analyzed in Nedic and Ozdaglar (2009). The push-sum scheme of Section 2, inspired by Hadjicostis et al. (2016), is used instead of the consensus scheme, which allows us to handle delays, asynchronicity, and message losses; this is similar to the approach taken in Nedic and Olshevsky (2015). We note that a new step-size strategy is used to handle asynchronicity: when a node wakes up, it takes steps with a step-size proportional to the sum of all the step-sizes during the period it slept. As far as we are aware, this idea is new.

We will be making the following assumption on the noise vectors.

Assumption 2 *$\boldsymbol{\varepsilon}_i$ is an independent random vector with bounded support, i.e., $\|\boldsymbol{\varepsilon}_i\| \le b_i$, $i = 1, \ldots, n$. Moreover, $\mathbb{E}[\boldsymbol{\varepsilon}_i] = \mathbf{0}$ and $\mathbb{E}[\|\boldsymbol{\varepsilon}_i\|^2] \le \sigma_i^2$.*

Next, we state and prove the main result of this paper, which states the linear convergence rate of Algorithm 3.

Theorem 15 *Suppose that:*

1. *Assumptions 1 and 2 hold.*
2. *Each objective function $f_i(\mathbf{z})$ is μ_i-strongly convex over $\mathbb{R}^d$.*
3. *The gradients of each $f_i(\mathbf{z})$ are L_i-Lipschitz continuous, i.e., for all $\mathbf{z}_1, \mathbf{z}_2 \in \mathbb{R}^d$,*

$$\|\mathbf{g}_i(\mathbf{z}_1) - \mathbf{g}_i(\mathbf{z}_2)\| \le L_i \|\mathbf{z}_1 - \mathbf{z}_2\|.$$

Then, the RASGP algorithm with the step-size $\alpha(k) = n/(\mu k)$ for $k \ge 1$ and $\alpha(0) = 0$, will converge to the unique optimum $\mathbf{z}^*$ with the following asymptotic rate: for all $i = 1, \ldots, n$, we have

$$\mathbb{E}\left[\|\mathbf{z}_i(k) - \mathbf{z}^*\|^2\right] \le \frac{\Gamma_u \sigma^2}{k\mu^2} + \mathcal{O}_k\left(\frac{1}{k^{1.5}}\right),$$

where $\sigma^2 := \sum_i \sigma_i^2$, $\mu = \sum_i \mu_i$.

Algorithm 3 Robust Asynchronous Stochastic Gradient-Push (RASGP)

```
Initialize the algorithm with $\mathbf{y}(0) = \mathbf{1}$, $\boldsymbol{\phi}_i^x(0) = \mathbf{0}$, $\phi_i^y(0) = 0$, $\kappa_i(0) = -1$, $\forall i \in \{1, \ldots, n\}$ and $\boldsymbol{\rho}_{ij}^x(0) = \mathbf{0}$, $\rho_{ij}^y(0) = 0$, $\kappa_{ij}(0) = -1$, $\forall (j, i) \in \mathcal{E}$.
At every iteration $k = 0, 1, 2, \ldots$, for every node $i$:
if node $i$ wakes up then
    $\beta_i(k) = \sum_{t=\kappa_i+1}^{k} \alpha(t)$;
    $\mathbf{x}_i \leftarrow \mathbf{x}_i - \beta_i(k)\hat{\mathbf{g}}_i(k)$;
    $\kappa_i \leftarrow k$;
    $\boldsymbol{\phi}_i^x \leftarrow \boldsymbol{\phi}_i^x + \frac{\mathbf{x}_i}{d_i^+ + 1}$, $\phi_i^y \leftarrow \phi_i^y + \frac{y_i}{d_i^+ + 1}$;
    $\mathbf{x}_i \leftarrow \frac{\mathbf{x}_i}{d_i^+ + 1}$, $y_i \leftarrow \frac{y_i}{d_i^+ + 1}$;
    Node $i$ broadcasts $(\boldsymbol{\phi}_i^x, \phi_i^y, \kappa_i)$ to its out-neighbors: $N_i^+$
    Processing the received messages
    for $(\boldsymbol{\phi}_j^x, \phi_j^y, \kappa_j')$ in the inbox do
        if $\kappa_j' > \kappa_{ij}$ then
            $\boldsymbol{\rho}_{ij}^{*x} \leftarrow \boldsymbol{\phi}_j^x$, $\rho_{ij}^{*y} \leftarrow \phi_j^y$;
            $\kappa_{ij} \leftarrow \kappa_j'$;
        end if
    end for
    $\mathbf{x}_i \leftarrow \mathbf{x}_i + \sum_{j \in N_i^-} \left(\boldsymbol{\rho}_{ij}^{*x} - \boldsymbol{\rho}_{ij}^x\right)$, $y_i \leftarrow y_i + \sum_{j \in N_i^-} \left(\rho_{ij}^{*y} - \rho_{ij}^y\right)$;
    $\boldsymbol{\rho}_{ij}^x \leftarrow \boldsymbol{\rho}_{ij}^{*x}$, $\rho_{ij}^y \leftarrow \rho_{ij}^{*y}$;
    $\mathbf{z}_i \leftarrow \frac{\mathbf{x}_i}{y_i}$;
end if
Other variables remain unchanged.
```

Remark 16 *We note that each agent stores variables $\mathbf{x}_i, y_i, \kappa_i, \mathbf{z}_i, \boldsymbol{\phi}_i^x, \phi_i^y$ and $\boldsymbol{\rho}_{ij}^x, \rho_{ij}^y, \kappa_{ij}$ for all in-neighbors $j \in N_i^-$. Hence, the memory requirement of the RASGP algorithm for each agent is $\mathcal{O}(d_i^-)$ for each agent i.*

We next turn to the proof of Theorem 15. First, we observe that Algorithm 3 is a specific case of multi-dimensional Perturbed Robust Asynchronous Push-Sum. In other words, each coordinate of vectors $\mathbf{x}_i$, $\mathbf{z}_i$, $\boldsymbol{\phi}_i^x$ and $\boldsymbol{\rho}_{ij}^x$ will experience an instance of Algorithm 2. Hence, there exists an augmented graph sequence $\{\mathcal{H}(k)\}$ where the Algorithm 3 is equivalent to perturbed push-sum consensus on $\mathcal{H}(k)$ where each agent $a_h \in \mathcal{V}_A$ holds vectors $\mathbf{x}_h$ and y_h. In other words, we will be able to apply Theorem 14 to analyze Algorithm 3.

Our first step is to show how to decouple the action of Algorithm 3 coordinate by coordinate. For each coordinate $1 \leq \ell \leq d$, let $\boldsymbol{\chi}^\ell \in \mathbb{R}^{n+m'}$ stack up the ℓth entries of x-values of all agents (virtual and non-virtual) in $\mathcal{V}_A$. Additionally, define $\boldsymbol{\Delta}^\ell(k) \in \mathbb{R}^{n+m'}$ to be the vector stacking up the ℓth entries of perturbations. i.e.,

$$[\boldsymbol{\Delta}^\ell(k)]_i := \begin{cases} -\beta_i(k)[\hat{\mathbf{g}}_i(k)]_\ell, & \text{if } i \in \mathcal{V}, \tau_i(k) = 1, \\ 0, & \text{otherwise.} \end{cases}$$

Then, by the definition of the algorithm, we have for all $\ell = 1, \dots, d$,

$$\begin{aligned}\boldsymbol{\chi}^{\ell}(k+1) &= \mathbf{M}(k)\left(\boldsymbol{\chi}^{\ell}(k) + \boldsymbol{\Delta}^{\ell}(k)\right),\\ \boldsymbol{\psi}(k+1) &= \mathbf{M}(k)\boldsymbol{\psi}(k).\end{aligned} \tag{28}$$

These equations write out the action of Algorithm 3 on a coordinate-by-coordinate basis.

In order to prove Theorem 15, we need a few tools and lemmas. As already mentioned, our first step will be to argue that Algorithm 3 converges by application of Theorem 14. This requires showing the boundedness of the perturbations $\boldsymbol{\Delta}^{\ell}(k)$, which, as we will show, reduces to showing the vectors $\mathbf{z}_i(k)$ are bounded. The following lemma will be useful to establish this boundedness.

Lemma 17 *(Nedic and Olshevsky, 2016, Lemma 3) Let $q : \mathbb{R}^d \to \mathbb{R}$ be a ν-strongly convex function with $\nu > 0$ which has Lipschitz gradients with constant L. let $\mathbf{v} \in \mathbb{R}^d$ and $\mathbf{u} \in \mathbb{R}^d$ defined by,*

$$\mathbf{u} = \mathbf{v} - \alpha(\nabla q(\mathbf{v}) + p(\mathbf{v})),$$

where $\alpha \in \left(0, \nu/8L^2\right]$ and $p : \mathbb{R}^d \to \mathbb{R}^d$ is a mapping such that,

$$\|p(\mathbf{v})\| \le c, \qquad \text{for all } \mathbf{v} \in \mathbb{R}^d.$$

Then, there exists a compact set $\mathcal{S} \subset \mathbb{R}^d$ and a scalar R such that,

$$\|\mathbf{u}\| \le \begin{cases} \|\mathbf{v}\|, & \text{for all } \mathbf{v} \notin \mathcal{S},\\ R, & \text{for all } \mathbf{v} \in \mathcal{S},\end{cases}$$

where,

$$\mathcal{S} := \{\mathbf{z} \,|\, q(\mathbf{z}) \le q(\mathbf{0}) + 2\frac{\nu}{8L^2}\left(\|q(\mathbf{0})\|^2 + c^2\right)\} \cup B\left(\mathbf{0}, \frac{4c}{\nu}\right),$$

$$R := \max_{\mathbf{z}\in\mathcal{S}}\{\|\mathbf{z}\| + \frac{\nu}{8L^2}\|\nabla q(\mathbf{z})\|\} + \frac{\nu c}{8L^2}.$$

We now argue that the iterates generated by Algorithm 3 are bounded.

Lemma 18 *The iterates $\mathbf{z}_i(k)$ generated by Algorithm 3 will remain bounded.*

Proof Let us adopt the notation $\boldsymbol{\psi}^-$ from previous sections and define $\mathbf{z}^{\ell}(k) := \boldsymbol{\chi}^{\ell}(k) \circ \boldsymbol{\psi}^-(k) \in \mathbb{R}^{n+m'}$. Moreover, adopt the notation $\mathbf{z}_h$ for virtual agent a_h, $h = n+1, \dots, n+m'$, as $\mathbf{z}_h(k) := \mathbf{x}_h(k)/\psi_h(k)$. Also define $\mathbf{u}^{\ell} \in \mathbb{R}^{n+m'}$ by

$$\mathbf{u}^{\ell}(k) := \boldsymbol{\chi}^{\ell}(k) + \boldsymbol{\Delta}^{\ell}(k).$$

Since the perturbations are only added to the non-virtual agents, which have strictly positive y-values, we conclude $[u^{\ell}(k)]_h = 0$ if $\psi_h(k) = 0$. Hence, the assumptions of Lemma 8 and Corollary 9 are satisfied. Adopting the definition of I^k and $\mathbf{P}(k)$ from previous sections, we get for $i \in I^{k+1}$,

$$[z^{\ell}(k+1)]_i = \sum_{j\in I^k} P_{ij}(k)\frac{[u^{\ell}(k)]_j}{\psi_j(k)}.$$

Combining the equation above for $\ell = 1, \dots, d$ we obtain:

$$\mathbf{z}_i(k+1) = \sum_{j \in I^k} P_{ij}(k) \frac{\mathbf{u}_j(k)}{\psi_j(k)}, \tag{29}$$

where $\mathbf{u}_j(k) \in \mathbb{R}^d$ is created by collecting the jth entries of all $\mathbf{u}^\ell(k)$, i.e.,

$$\mathbf{u}_i(k) = \begin{cases} \mathbf{x}_i(k) - \beta_i(k)\hat{\mathbf{g}}_i(k), & \text{if } i \in \mathcal{V} \text{ and } \tau_i(k) = 1, \\ \mathbf{x}_i(k), & \text{otherwise.} \end{cases}$$

Now consider each term on the right hand side of (29) for $j \in I^k$. Suppose $j \leq n$ and $\tau_j(k) = 1$, then we have:

$$\frac{\mathbf{u}_j(k)}{y_j(k)} = \mathbf{z}_j(k) - \frac{\beta_j(k)}{y_j(k)}(\nabla f_j(\mathbf{z}_j(k)) + \boldsymbol{\varepsilon}_j(k)).$$

Since $\lim_{k\to\infty} \alpha(k) = 0$ and $k - \kappa_i(k) \leq \Gamma_u$, $\lim_{k\to\infty} \beta_j(k) = 0$. Moreover, by Lemma 11, $y_j(k)$ is bounded below; thus, $\lim_{k\to\infty} \beta_j(k)/y_j(k) = 0$ and there exists k_j such that for $k \geq k_j$, $\beta_j(k)/y_j(k) \in \left(0, \mu_j/8L_j^2\right]$. Applying Lemma 17, it follows that for each j there exists a compact set $\mathcal{S}_j$ and a scalar R_j such that for $k \geq k_j$, if $\tau_j(k) = 1$,

$$\left\| \frac{\mathbf{u}_j(k)}{y_j(k)} \right\| \leq \begin{cases} \|\mathbf{z}_j(k)\|, & \text{if } \mathbf{z}_j(k) \notin \mathcal{S}_j, \\ R_j, & \text{if } \mathbf{z}_j(k) \in \mathcal{S}_j. \end{cases} \tag{30}$$

Moreover, if $\tau_j(k) = 0$ or $j > n$ we have,

$$\frac{\mathbf{u}_j(k)}{y_j(k)} = \mathbf{z}_j(k). \tag{31}$$

Let $k_z := \max_i k_i$. Using mathematical induction, we will show that for all $k \geq k_z$:

$$\max_{i \in I^k} \|\mathbf{z}_i(k)\| \leq \bar{R}, \tag{32}$$

where $\bar{R} := \max\{\max_i R_i, \max_{j \in I^{k_z}} \|\mathbf{z}_j(k_z)\|\}$. Equation (32) holds for $k = k_z$. Suppose it is true for some $k \geq k_z$. Then by (30) and (31) we have,

$$\left\| \frac{\mathbf{u}_i(k)}{y_i(k)} \right\| \leq \max\{R_i, \|\mathbf{z}_i(k)\|\} \leq \bar{R}. \tag{33}$$

Also by (29), for $i \in I^{k+1}$, $\mathbf{z}_i(k+1)$ is a convex combination of $\mathbf{u}_j(k)/y_j(k)$'s, where $j \in I^k$. Hence,

$$\|\mathbf{z}_i(k+1)\| \leq \sum_{j \in I^k} P_{ij} \left\| \frac{\mathbf{u}_j(k)}{\psi_j(k)} \right\| \leq \bar{R}.$$

Define $B_z := \max\{\bar{R}, \max_{i \in I^k, k < k_z} \|\mathbf{z}_i(k)\|\}$ and we have $\|\mathbf{z}_i(k)\| \leq B_z$, $\forall k \geq 0$. ■

We next explore a convenient way to rewrite Algorithm 3. Let us introduce the quantity $\mathbf{w}_i(k)$, which can be interpreted as the x-value of agent i, if it performed a gradient step at every iteration, even when asleep:

$$\mathbf{w}_i(k) := \begin{cases} \mathbf{x}_i(k) - \left(\sum_{t=\kappa_i(k)+1}^{k-1} \alpha(t)\right) \mathbf{g}_i(k), & \text{if } i \in \mathcal{V}, \\ \mathbf{x}_i(k), & \text{otherwise.} \end{cases} \tag{34}$$

Also, define $\mathbf{w}^\ell \in \mathbb{R}^{n+m'}$ by collecting the ℓth dimension of all $\mathbf{w}_i$'s and $\bar{\mathbf{w}}(k) := (\sum_{i=1}^{n+m'} \mathbf{w}_i(k))/n$. Moreover, define $\mathbf{g}^\ell \in \mathbb{R}^{n+m'}$ by collecting the ℓth value of gradients of all agents (0 for virtual agents), i.e.,

$$[\mathbf{g}^\ell(k)]_i = \begin{cases} [\mathbf{g}_i(k)]_\ell, & \text{if } i \in \mathcal{V}, \\ 0, & \text{otherwise.} \end{cases}$$

Additionally, define $\hat{\boldsymbol{\varepsilon}}_i(k) \in \mathbb{R}^d$ as the noise injected to the system at time k by agent i, i.e.,

$$\hat{\boldsymbol{\varepsilon}}_i(k) = \begin{cases} \beta_i(k)\boldsymbol{\varepsilon}_i(k), & \text{if } i \in \mathcal{V} \text{ and } \tau_i(k) = 1, \\ \mathbf{0}, & \text{otherwise,} \end{cases}$$

and $\hat{\boldsymbol{\varepsilon}}^\ell(k) \in \mathbb{R}^{n+m'}$ as the vector collecting the ℓth values of all $\hat{\boldsymbol{\varepsilon}}_i(k)$'s.

We then have the following lemma.

Lemma 19

$$\mathbf{w}^\ell(k+1) = \mathbf{M}(k) \left(\mathbf{w}^\ell(k) - \alpha(k)\mathbf{g}^\ell(k) - \hat{\boldsymbol{\varepsilon}}^\ell\right). \tag{35}$$

Proof We consider two cases:

- If $\tau_i(k) = 0$, then (35) reduces to $\mathbf{w}_i(k+1) = \mathbf{w}_i(k) - \alpha(k)\mathbf{g}_i(k)$; noting that, because node i did not update at time k we have that $\mathbf{g}_i(k) = \mathbf{g}_i(k+1)$ and this is the correct update.
- For all other nodes (i.e., for both virtual nodes and nodes with $\tau_i(k) = 1$), we have $[\mathbf{w}^\ell(k) - \alpha(k)\hat{\mathbf{g}}^\ell(k) - \hat{\boldsymbol{\varepsilon}}^\ell(k)]_i = [\boldsymbol{\chi}^\ell(k) + \Delta^\ell(k)]_i$ in (28). Since $\boldsymbol{\chi}^\ell(k+1) = \mathbf{M}(k)(\boldsymbol{\chi}^\ell(k) + \boldsymbol{\Delta}^\ell(k))$ and, using the definition of $\mathbf{w}_i(k)$, we have that for these nodes,

$$\mathbf{w}_i(k+1) = \mathbf{x}_i(k+1);$$

  (28) implies the conclusion.

■

This lemma allows us to straightforwardly analyze how the average of $\mathbf{w}(k)$ evolves. Indeed, summing all the elements of (35) and dividing by n for each $\ell = 1, \ldots, d$ we obtain,

$$\begin{aligned}\bar{\mathbf{w}}(k+1) &= \bar{\mathbf{w}}(k) - \frac{\alpha(k)}{n}\sum_{i=1}^{n}\mathbf{g}_i(k) - \frac{1}{n}\sum_{i=1}^{n}\hat{\boldsymbol{\varepsilon}}_i(k)\\ &= \bar{\mathbf{w}}(k) - \frac{\alpha(k)}{n}\sum_{i=1}^{n}\nabla f_i(\bar{\mathbf{w}}(k)) - \frac{1}{n}\sum_{i=1}^{n}\hat{\boldsymbol{\varepsilon}}_i(k) - \frac{\alpha(k)}{n}\sum_{i=1}^{n}\left(\mathbf{g}_i(k) - \nabla f_i(\bar{\mathbf{w}}(k))\right).\end{aligned} \tag{36}$$

We next give a sequence of lemmas to the effect that all the quantities generated by the algorithm are close to each other over time. Define,

$$\bar{\mathbf{x}}(k) = \frac{1}{n}\sum_{a_h\in\mathcal{V}_A}\mathbf{x}_h(k).$$

where, recall, $\mathcal{V}_A$ is our notation for all the nodes in the augmented graph (i.e., including virtual nodes). Moreover, we will extend the definition of $\beta_i(k)$ from Line 4 of Algorithm 3 to *all* k via the same formula $\beta_i(k) := \sum_{t=\kappa_i(k)+1}^{k}\alpha(t)$. Our first lemma will show that each $\mathbf{z}_i(k)$ closely tracks $\bar{\mathbf{x}}(k)$.

Lemma 20 *Using Algorithm 3 with $\alpha(k) = n/(k\mu)$, under the assumptions of Theorem 15, we have for each i, $\|\mathbf{z}_i(k+1) - \bar{\mathbf{x}}(k+1)\| = \mathcal{O}_k(1/k)$.*

Proof By Theorem 14(a) we have for each ℓ,

$$\left|[\mathbf{z}^\ell(k+1)]_i - \frac{\mathbf{1}^\top\boldsymbol{\chi}^\ell(k+1)}{n}\right| \le \delta\lambda^k\|\boldsymbol{\chi}^\ell(0)\|_1 + \sum_{t=1}^{k}\delta\lambda^{k-t}\|\boldsymbol{\Delta}^\ell(t)\|_1.$$

Summing the above inequality for $\ell = 1,\ldots,d$ we obtain,

$$\|\mathbf{z}_i(k+1) - \bar{\mathbf{x}}(k+1)\|_1 \le \sum_{j=1}^{n}\Big(\delta\lambda^k\|\mathbf{x}_j(0)\|_1 + \sum_{t=1}^{k}\delta\lambda^{k-t}\beta_i(t)\tau_i(t)\|\hat{\mathbf{g}}_j(t)\|_1\Big).$$

Moreover,

$$\beta_i(k) = \sum_{t=\kappa_i(k)+1}^{k}\frac{n}{\mu t} \le \frac{n}{\mu}\left(\frac{k-\kappa_i(k)}{\kappa_i(k)+1}\right). \tag{37}$$

But

$$\kappa_i(k) < k \le \kappa_i(k) + \Gamma_u.$$

Since $\Gamma_u \ge 1$, we obtain

$$k \le (\kappa_i(k)+1)\Gamma_u,$$

or,

$$\frac{1}{\kappa_i(k)+1} \le \frac{\Gamma_u}{k}.$$

Thus, from (37) we have,

$$\beta_i(k) \leq \frac{n\Gamma_u^2}{\mu k}. \tag{38}$$

Define

$$M_j := \max_{\|\mathbf{z}\| \leq B_z} \|\mathbf{g}_j(\mathbf{z})\|_1, \tag{39}$$

and observe that M_j is finite by Lemma 18. Also $\tau_j(k) \leq 1$. We obtain,

$$\|\mathbf{z}_i(k+1) - \bar{\mathbf{x}}(k+1)\|_1 \leq \sum_{j=1}^{n} \Big(\delta\lambda^k \|\mathbf{x}_j(0)\|_1 + \sum_{t=1}^{k} \delta\lambda^{k-t} \frac{n\Gamma_u^2}{\mu t}(M_j + b_j)\Big).$$

Let RHS denote the right hand side of the relation above. We have,

$$\begin{aligned} RHS &= \sum_{j=1}^{n} \left(\delta\lambda^k \|\mathbf{x}_j(0)\|_1 + \frac{\delta n\Gamma_u^2}{\mu}(M_j + b_j)\Big(\sum_{t=1}^{\lfloor\frac{k}{2}\rfloor} \frac{\lambda^{k-t}}{t} + \sum_{t=\lfloor\frac{k}{2}\rfloor+1}^{k} \frac{\lambda^{k-t}}{t}\Big)\right) \\ &\leq \sum_{j=1}^{n} \left(\delta\lambda^k \|\mathbf{x}_j(0)\|_1 + \frac{\delta n\Gamma_u^2}{\mu}(M_j + b_j)\left(\frac{k}{2}\lambda^{\frac{k}{2}} + \frac{2}{(1-\lambda)k}\right)\right) = \mathcal{O}_k\left(\frac{1}{k}\right), \end{aligned}$$

where we used the following relations,

$$\sum_{t=1}^{\lfloor\frac{k}{2}\rfloor} \frac{\lambda^{k-t}}{t} \leq \lfloor\frac{k}{2}\rfloor \lambda^{k-\lfloor\frac{k}{2}\rfloor} \leq \frac{k}{2}\lambda^{\frac{k}{2}},$$

$$\sum_{t=\lfloor\frac{k}{2}\rfloor+1}^{k} \frac{\lambda^{k-t}}{t} \leq \sum_{t=0}^{\lceil\frac{k}{2}\rceil-1} \frac{\lambda^t}{\lfloor\frac{k}{2}\rfloor+1} \leq \frac{2}{(1-\lambda)k}.$$

Finally, $\|\mathbf{v}\|_2 \leq \|\mathbf{v}\|_1$ for all vectors $\mathbf{v}$, completes the proof. ■

An immediate consequence of this lemma is that the quantities $\bar{\mathbf{x}}(k)$ and $\bar{\mathbf{w}}(k)$ are close to each other.

Lemma 21 *Using Algorithm 3 with $\alpha(k) = n/(k\mu)$, under the assumptions of Theorem 15, we have, $\|\bar{\mathbf{x}}(k) - \bar{\mathbf{w}}(k)\| = \mathcal{O}_k(1/k)$.*

Proof By definition of $\bar{\mathbf{w}}$ we have,

$$\bar{\mathbf{x}}(k) - \bar{\mathbf{w}}(k) = \frac{1}{n}\sum_{i=1}^{n} \left(\sum_{t=\kappa_i(k)+1}^{k-1} \alpha(t)\right) \mathbf{g}_i(k).$$

Using (38) we have,

$$\|\bar{\mathbf{x}}(k) - \bar{\mathbf{w}}(k)\| \leq \frac{1}{n}\sum_{i=1}^{n} \beta_i(k) M_i \leq \sum_{i=1}^{n} \frac{\Gamma_u^2 M_i}{n\mu k} = \mathcal{O}_k\left(\frac{1}{k}\right),$$

where M_i was defined through (39). ■

We next remark on a couple of implications of the past series of lemmas.

Corollary 22 *We have* $\|\mathbf{z}_i(k) - \bar{\mathbf{w}}(k)\| = \mathcal{O}_k(1/k)$.

Lemma 23 $\|\mathbf{g}_i(k) - \nabla f_i(\bar{\mathbf{w}}(k))\| = \mathcal{O}_k(1/k)$.

Proof Since ∇f_i is L_i-Lipschitz, we have,

$$\|\mathbf{g}_i(k) - \nabla f_i(\bar{\mathbf{w}}(k))\| \leq L_i \|\mathbf{z}_i(k) - \bar{\mathbf{w}}(k)\|.$$

Using Corollary 22, the lemma is proved. ■

We are now in a position to rewrite Algorithm 3 as a sort of perturbed gradient descent. Let us define,

$$\boldsymbol{\eta}(k) := \frac{1}{\mu k} \sum_{i=1}^{n} \left(\mathbf{g}_i(k) - \nabla f_i(\bar{\mathbf{w}}(k)) \right).$$

By Lemma 23, $\boldsymbol{\eta}(k) = \mathcal{O}_k(1/k^2)$. Therefore, there exists B_η such that $\boldsymbol{\eta}(k) \leq B_\eta / k^2$ for all $k \geq 1$.

By (36) we have,

$$\bar{\mathbf{w}}(k+1) = \bar{\mathbf{w}}(k) - \frac{1}{\mu k} \nabla F(\bar{\mathbf{w}}(k)) - \bar{\boldsymbol{\varepsilon}}(k) - \boldsymbol{\eta}(k), \tag{40}$$

where

- The function $F := \sum_{i=1}^{n} f_i \in \mathbb{R}^d \to \mathbb{R}$ is μ-strongly-convex with L-Lipschitz gradient, where $L := \sum_{i=1}^{n} L_i$.
- The noise $\bar{\boldsymbol{\varepsilon}}(k) := (\sum_{i=1}^{n} \hat{\boldsymbol{\varepsilon}}_i(k))/n$ is bounded (i.e., $\bar{\boldsymbol{\varepsilon}}(k) \in B(0, r_e)$, with probability one, where $r_e := (\Gamma_u / \mu) \sum_j b_j)$, and $\mathbb{E}[\bar{\boldsymbol{\varepsilon}}(k)] = \mathbf{0}$.

In other words, with the exception of the $\boldsymbol{\eta}(k)$ term, what we have is exactly a stochastic gradient descent method on the function $F(\cdot)$.

The following lemmas bound $\bar{\boldsymbol{\varepsilon}}(k)$. Let us define $\nu_i(k) = k - \kappa_i(k)$ as the number of iterations agent i has skipped since it's last update. By Assumption 1, $\nu_i(k) \leq \Gamma_u$.

Lemma 24 *We have* $\beta_i(k) = \mathcal{O}_k(1/k)$, $\forall i$. *Moreover,*

$$\beta_i(k) \leq \frac{n \nu_i(k)}{\mu k} + \mathcal{O}_k(k^{-2}).$$

Proof Since $\nu_i(k) \leq \Gamma_u, \forall i$, we have for $\kappa_i(k) \geq 1$,

$$\begin{aligned} \beta_i(k) &= \sum_{t=\kappa_i(k)+1}^{k} \frac{n}{\mu t} \leq \frac{n}{\mu} \ln\left(\frac{k}{\kappa_i(k)}\right) \leq \frac{n}{\mu} \ln\left(\frac{k}{k - \nu_i(k)}\right) \\ &= \frac{n}{\mu} \ln\left(1 + \frac{\nu_i(k)}{k - \nu_i(k)}\right) \leq \frac{n \nu_i(k)}{\mu(k - \nu_i(k))} = \frac{n \nu_i(k)}{\mu k} + \mathcal{O}_k(k^{-2}). \end{aligned}$$

■

Corollary 25 *$\mu k\|\bar{\boldsymbol{\varepsilon}}(k)\|$ is bounded.*

Lemma 26 *There exists $B_\epsilon > 0$ such that We have,*

$$\mathbb{E}[\|\bar{\boldsymbol{\varepsilon}}(k)\|^2] \le \frac{\Gamma_u^2}{\mu^2 k^2}\sigma^2 + \frac{B_\epsilon}{k^4}.$$

Proof Using Lemma 24, we have for $k > \Gamma_u$,

$$\begin{aligned}\mathbb{E}[\|\bar{\boldsymbol{\varepsilon}}(k)\|^2] &= \mathbb{E}[\|\frac{1}{n}\sum_{i=1}^n \beta_i(k)\boldsymbol{\varepsilon}_i(k)\tau_i(k)\|^2] = \frac{1}{n^2}\sum_{i=1}^n \beta_i^2(k)\mathbb{E}[\|\boldsymbol{\varepsilon}_i(k)\|^2] \\ &\le \frac{1}{n^2}\sum_{i=1}^n \beta_i^2(k)\sigma_i^2 \le \frac{\Gamma_u^2}{\mu^2 k^2}\sigma^2 + \mathcal{O}_k(k^{-4}),\end{aligned}$$

where the second equality is the result of the noise terms being independent and zero-mean. ■

Our next observation is a technical lemma which is essentially a rephrasing of Lemma 17 above.

Lemma 27 *There exists a constant B_w and time k_w such that $\|\bar{\mathbf{w}}(k)\| \le B_w$ with probability one, for $k \ge k_w$.*

Proof We have

$$\bar{\mathbf{w}}(k+1) = \bar{\mathbf{w}}(k) - \frac{1}{\mu k}\left[\nabla F(\bar{\mathbf{w}}(k)) + \mu k\left(\bar{\boldsymbol{\varepsilon}}(k) + \boldsymbol{\eta}(k)\right)\right],$$

where $\mu k\|\bar{\boldsymbol{\varepsilon}}(k) + \boldsymbol{\eta}(k)\|$ is bounded. Moreover, there exists k_w such that for $k \ge k_w$, $\frac{1}{\mu k} \in \left(0, \mu/8L^2\right]$. Therefore, by Lemma 17 there exists a compact set $\mathcal{S}_w$ and a scalar $R_w > 0$ such that for $k \ge k_w$,

$$\|\bar{\mathbf{w}}(k+1)\| \le \begin{cases}\|\bar{\mathbf{w}}(k)\|, & \text{for } \bar{\mathbf{w}} \notin \mathcal{S}_w, \\ R_w, & \text{for } \bar{\mathbf{w}} \in \mathcal{S}_w.\end{cases}$$

Therefore, setting $B_w := \max\{R_w, \|\bar{\mathbf{w}}(k_w)\|\}$ will complete the proof. ■

As a consequence of this lemma, because $\|\boldsymbol{\eta}(k)\|_2 \le B_\eta$, this lemma implies there is a constant B_1 such that for $k \ge k_w$,

$$\left\|\bar{\mathbf{w}}(k) - \mathbf{z}^* - \frac{1}{\mu k}\nabla F(\bar{\mathbf{w}}(k)) - \bar{\boldsymbol{\varepsilon}}(k)\right\| \le B_1, \tag{41}$$

with probability one. This now puts us in a position to show that $\bar{\mathbf{w}}(k)$ converges in mean square to the optimal solution.

Lemma 28 $\mathbb{E}[\|\bar{\mathbf{w}}(k) - \mathbf{z}^*\|^2] \to 0$.

Proof Using the definition of k_w from Lemma 27, we have that for $k \geq k_w$,

$$\mathbb{E}[\|\bar{\mathbf{w}}(k+1) - \mathbf{z}^*\|^2] \leq \mathbb{E}\Big[\|\bar{\mathbf{w}}(k) - \mathbf{z}^* - \frac{1}{\mu k}\nabla F(\bar{\mathbf{w}}(k)) - \bar{\varepsilon}(k)\|^2 \\ + 2\|\boldsymbol{\eta}(k)\|\|\bar{\mathbf{w}}(k) - \mathbf{z}^* - \frac{1}{\mu k}\nabla F(\bar{\mathbf{w}}(k)) - \bar{\varepsilon}(k)\| + \|\boldsymbol{\eta}(k)\|^2\Big].$$

We will bound each of the terms on the right. We begin with the easiest one, which is the last one:

$$\|\boldsymbol{\eta}(k)\|^2 \leq \frac{B_\eta^2}{k^4}. \tag{42}$$

The middle term is bounded as

$$2\|\boldsymbol{\eta}(k)\|\|\bar{\mathbf{w}}(k) - \mathbf{z}^* - \frac{1}{\mu k}\nabla F(\bar{\mathbf{w}}(k)) - \bar{\varepsilon}(k)\| \leq \frac{2B_\eta B_1}{k^2}, \tag{43}$$

where we used (41).

Finally, we turn to the first term which we denote by T_1:

$$T_1 \leq \mathbb{E}\|\bar{\mathbf{w}}(k) - \mathbf{z}^*\|^2 - \frac{2}{\mu k}\mathbb{E}[\nabla F(\bar{\mathbf{w}}(k))^\top(\bar{\mathbf{w}}(k) - \mathbf{z}^*)] \\ + \frac{L^2}{\mu^2 k^2}\mathbb{E}[\|\bar{\mathbf{w}}(k) - \mathbf{z}^*\|^2] + \mathbb{E}[\|\bar{\varepsilon}(k)\|^2],$$

where we used the usual inequality $\|\nabla F(\bar{\mathbf{w}}(k))\|^2 \leq L^2\|\bar{\mathbf{w}}(k) - \mathbf{z}^*\|^2$ which follows from $\nabla F(\cdot)$ being L-Lipschitz. Now, using the standard inequality

$$\nabla F(\bar{\mathbf{w}}(k))^T(\bar{\mathbf{w}}(k) - \mathbf{z}^*) \geq F(\bar{\mathbf{w}}(k)) - F(\mathbf{z}^*) + \frac{\mu}{2}\|\bar{\mathbf{w}}(k) - \mathbf{z}^*\|^2 \\ \geq \mu\|\bar{\mathbf{w}}(k) - \mathbf{z}^*\|^2,$$

and Lemma 26 we obtain,

$$T_1 \leq \left(1 - \frac{2}{k} + \frac{L^2}{\mu^2 k^2}\right)\mathbb{E}[\|\bar{\mathbf{w}}(k) - \mathbf{z}^*\|^2] + \frac{\Gamma_u^2}{\mu^2 k^2}\sigma^2 + \frac{B_\epsilon}{k^4}. \tag{44}$$

Now putting together (42), (43), and (44), we get,

$$\mathbb{E}[\|\bar{\mathbf{w}}(k+1) - \mathbf{z}^*\|^2] \leq \left(1 - \frac{2}{k} + \frac{L^2}{\mu^2 k^2}\right)\mathbb{E}[\|\bar{\mathbf{w}}(k) - x^*\|^2] + \frac{\Gamma_u^2\sigma^2}{\mu^2 k^2} + \frac{2B_\eta B_1}{k^2} + \frac{B_\eta^2 + B_\epsilon}{k^4}.$$

For large enough k, we can bound the inequality above as,

$$\mathbb{E}[\|\bar{\mathbf{w}}(k+1) - \mathbf{z}^*\|^2] \leq \left(1 - \frac{1.5}{k}\right)\mathbb{E}[\|\bar{\mathbf{w}}(k) - \mathbf{z}^*\|^2] + \frac{B_2}{k^2}, \tag{45}$$

where $B_2 = \Gamma_u^2\sigma^2/\mu^2 + 2B_\eta B_1 + B_\eta^2 + B_\epsilon$. Using Lemma 29, stated next, we conclude $\mathbb{E}[\|\bar{\mathbf{w}}(k) - \mathbf{z}^*\|^2] \to 0$. ■

Lemma 29 *Let $a > 1, b \geq 0$ and $\{x_t\}$ be a non-negative sequence which satisfies,*

$$x_{t+1} \leq \left(1 - \frac{a}{t}\right) x_t + \frac{b}{t^2}, \qquad \textit{for } t \geq t' > 0.$$

Then for all $t \geq t'$ we have,

$$x_t \leq \frac{m}{t},$$

where $m := \max\{t' x_{t'}, b/(a-1)\}$.

This lemma is stated and proved for $t' = 1$ in (Rakhlin et al., 2012, Lemma 3), and the case of general t' follows immediately.

We are almost ready to complete the proof of Theorem 15; all that is needed is to refine the convergence rate of $\bar{\mathbf{w}}(k)$ to x^*. Now as a consequence of (45) and Lemma 29, we may use the inequality $\mathbb{E}[|X|] \leq \sqrt{\mathbb{E}[X^2]}$ to obtain that

$$\mathbb{E}[\|\bar{\mathbf{w}}(k) - \mathbf{z}^*\|] = \mathcal{O}_k\left(\frac{1}{\sqrt{k}}\right). \tag{46}$$

Furthermore, by the finite support of $\mu k \bar{\varepsilon}(k)$, by Corollary 25, we also have that

$$\mathbb{E}[\|\bar{\mathbf{w}}(k) - \mathbf{z}^* - \frac{1}{\mu k}\nabla F(\bar{\mathbf{w}}(k)) - \bar{\varepsilon}(k)\|] = \mathcal{O}_k\left(\frac{1}{\sqrt{k}}\right). \tag{47}$$

We now use these observations to provide a proof of our main result.

Proof of Theorem 15 Essentially, we rewrite the proof of Lemma 28, but now using the fact that $\mathbb{E}[\|\bar{\mathbf{w}}(k) - \mathbf{z}^*\|] = \mathcal{O}_k(1/\sqrt{k})$ from (46). This allows us to make two modification to the arguments of that lemma. First, we can now replace (43) by

$$\mathbb{E}[2\|\boldsymbol{\eta}(k)\|\|\bar{\mathbf{w}}(k) - \mathbf{z}^* - \frac{1}{\mu k}\nabla F(\bar{\mathbf{w}}(k)) - \bar{\varepsilon}(k)\|] \leq \frac{2B_\eta}{k^2}\mathcal{O}_k\left(\frac{1}{\sqrt{k}}\right), \tag{48}$$

where we used (47). Second, putting together (42), (48), and (44), we obtain:

$$\mathbb{E}[\|\bar{\mathbf{w}}(k+1) - \mathbf{z}^*\|^2] \leq \left(1 - \frac{2}{k} + \frac{L^2}{\mu^2 k^2}\right)\mathbb{E}[\|\bar{\mathbf{w}}(k) - \mathbf{z}^*\|^2] + \mathbb{E}[\|\bar{\varepsilon}(k)\|^2] + \frac{B_\eta^2}{k^4} + \frac{2B_\eta}{k^2}\mathcal{O}_k\left(\frac{1}{\sqrt{k}}\right).$$

which, again using the fact that $\mathbb{E}[\|\bar{\mathbf{w}}(k) - \mathbf{z}^*\|^2] = \mathcal{O}_k(1/\sqrt{k})$, we simply rewrite as,

$$\mathbb{E}[\|\bar{\mathbf{w}}(k+1) - \mathbf{z}^*\|^2] \leq \left(1 - \frac{2}{k}\right)\mathbb{E}[\|\bar{\mathbf{w}}(k) - \mathbf{z}^*\|^2] + \mathbb{E}[\|\bar{\varepsilon}(k)\|^2] + \mathcal{O}_k\left(\frac{1}{k^{2.5}}\right).$$

To save space, let us define $a_k := \mathbb{E}[\|\bar{\mathbf{w}}(k) - \mathbf{z}^*\|^2]$. Multiplying both sides of relation above by k^2 we obtain,

$$a_{k+1}k^2 \leq a_k\left(1 - \frac{2}{k}\right)k^2 + \mathbb{E}[\|\bar{\varepsilon}(k)\|^2]k^2 + \mathcal{O}_k(k^{-0.5}).$$

Note that,

$$\left(1 - \frac{2}{k}\right) k^2 = k^2 - 2k < (k-1)^2.$$

Thus,

$$a_{k+1} k^2 \leq a_k (k-1)^2 + \mathbb{E}[\|\bar{\varepsilon}(k)\|^2] k^2 + \mathcal{O}_k(k^{-0.5}).$$

Summing the relation above for $k = 0, \ldots, T$ implies,

$$a_{T+1} T^2 \leq \sum_{k=0}^{T} \mathbb{E}[\|\bar{\varepsilon}(k)\|^2] k^2 + \mathcal{O}_T(T^{0.5}).$$

Now, let us estimate the first term on the right hand side of relation above,

$$\sum_{k=0}^{T} \mathbb{E}[\|\bar{\varepsilon}(k)\|^2] k^2 \leq \sum_{k=0}^{T} \sum_{i=1}^{n} \frac{\beta_i^2(k)}{n^2} \sigma_i^2 \tau_i(k) k^2 = \sum_{i=1}^{n} \frac{\sigma_i^2}{\mu^2} \sum_{k=0}^{T} \nu_i(k)^2 \tau_i(k) + \mathcal{O}_T(T^{-1}),$$

where we used Lemma 24 in the last equality. Define $t_i(j)$ as the j'th time agent i has woken up, and set $t_i(0) = -1$. Then we can rewrite the relation above as,

$$\sum_{k=0}^{T} \nu_i(k)^2 \tau_i(k) = \sum_{j=1}^{t_i(j) \leq T} (t_i(j) - t_i(j-1))^2 \leq \sum_{j=1}^{t_i(j) \leq T} \Gamma_u (t_i(j) - t_i(j-1)) \leq \Gamma_u (T+1).$$

Combining relations above and then dividing both sides by T^2 we obtain,

$$a_{T+1} \leq \frac{\Gamma_u \sigma^2}{\mu^2 T} + \mathcal{O}_T(T^{-1.5}). \tag{49}$$

We next argue that the same guarantee holds for every $\mathbf{z}_i(k)$. Indeed, for each $i = 1, \ldots, m$,

$$\begin{aligned} \|\mathbf{z}_i(k) - \mathbf{z}^*\|^2 &= \|\mathbf{z}_i(k) - \bar{\mathbf{w}}(k) + \bar{\mathbf{w}}(k) - \mathbf{z}^*\|^2 \\ &= \|\mathbf{z}_i(k) - \bar{\mathbf{w}}(k)\|^2 + 2\|\mathbf{z}_i(k) - \bar{\mathbf{w}}(k)\| \|\bar{\mathbf{w}}(k) - \mathbf{z}^*\| + \|\bar{\mathbf{w}}(k) - \mathbf{z}^*\|^2. \end{aligned}$$

Now from Corollary 22, we know that with probability one, $\|\mathbf{z}_i(k) - \bar{\mathbf{w}}(k)\|_2 = \mathcal{O}_k(1/k)$. Taking expectation of both sides and using (49) along with the usual bound $\mathbb{E}[|X|] \leq \sqrt{\mathbb{E}[X^2]}$, we have

$$\mathbb{E}[\|\mathbf{z}_i(k) - \mathbf{z}^*\|^2] = \mathcal{O}_k\left(\frac{1}{k^2}\right) + \mathcal{O}_k\left(\frac{1}{k^{1.5}}\right) + \mathbb{E}[\|\bar{\mathbf{w}}(k) - \mathbf{z}^*\|^2].$$

Putting this together with (49) completes the proof. ■

3.1. Time-Varying Graphs

We remark that Theorems 6, 14 and 15 all extend verbatim to the case of time-varying graphs with no message losses. Indeed, only one problem appears in extending the proofs in this paper to time-varying graphs: a node i may send a message to node j; that message will be lost; and afterwards node i never sends anything to node j again. In this case, Lemmas 7 and 11 do not hold. Indeed, examining Lemma 11, we observe what can very well happen is that all of $\chi_i(k)$ and $\psi_i(k)$ are "lost" over time into messages that never arrive. However, as long as no messages are lost, the proofs in this paper extend to the time-varying case verbatim. On a technical level, the results still hold if $\mathbf{u}_{ij}^x(k) = \mathbf{0}, u_{ij}^y(k) = 0$ (virtual node $c_{ij} \in \mathcal{V}_A$ holds no lost message), when link (i, j) is removed from the network at time k, and the graph $\mathcal{G}$ stays strongly connected (or B-connected, i.e., there exists a positive integer B such that the union of every B consecutive graphs is strongly connected).

3.2. On the Bounds for Delays, Asynchrony, and Message Losses

It is natural to what extent the assumption of finite upper bounds on delays, asynchrony, and message losses are really necessary. A natural example which falls outside our framework is a fixed graph G, where, at each time step, every link in G appears with probability 1/2. A more general model might involve a different probability p_e of failure for each edge e.

We observe that our result can already handle this case in the following manner. For simplicity, let us stick with the scenario where every link appears with probability 1/2. Then the probability that, after time t, some link has not appeared is at most $m(1/2)^t$, where m is the number of edges in G. This implies that if we choose $B = O(\log(mnT))$, then with high probability, the sequence of graphs $G_1, \ldots, G_T$ is B-connected.

Thus our theorem applies to this case, albeit at the expense of some logarithmic factors due to the choice of B. We remark that it is possible to get rid of these factors by directly analyzing the decrease in $E[||z(t) - z^*||_2^2]$ coming from the random choice of graph G. Since our arguments are already quite lengthy, we do not pursue this generalization here, and refer the reader to Lobel and Ozdaglar (2010); Srivastava and Nedic (2011) where similar arguments have been made.

4. Numerical Simulations

4.1. Setup

In this section, we simulate the RASGP algorithm on two classes of graphs, namely, random directed graphs and bidirectional cycle graphs. The main objective function is chosen to be a strongly convex and smooth Support Vector Machine (SVM), i.e. $F(\boldsymbol{\omega}, \gamma) = \frac{1}{2}\left(\|\boldsymbol{\omega}\|^2 + \gamma^2\right) + C_N \sum_{j=1}^N h(b_j(\mathbf{A}_j^\top \boldsymbol{\omega} + \gamma))$ where $\boldsymbol{\omega} \in \mathbb{R}^{d-1}$ and $\gamma \in \mathbb{R}$ are the optimization variables, and $\mathbf{A}_j \in \mathbb{R}^{d-1}, b_j \in \{-1, +1\}$, $j = 1, \ldots, N$, are the data points and their labels, respectively. The coefficient $C_N \in \mathbb{R}$ penalizes the points outside of the soft margin. We set $C_N = c/N, c = 500$ in our simulations, which depends on the total number of data points. Here, $h : \mathbb{R} \to \mathbb{R}$ is the smoothed hinge loss, initially introduced in Rennie and Srebro

(2005), defined as follows:

$$h(\xi) = \begin{cases} -0.5 - \xi, & \text{if } \xi < 0, \\ 0.5(1-\xi)^2, & \text{if } 0 \leq \xi < 1, \\ 0, & \text{if } 1 \leq \xi. \end{cases}$$

To solve this problem in a distributed way, we suppose all data points are spread among agents. Hence, the local objective functions are $f_i(\boldsymbol{\omega}_i, \gamma_i) = \frac{1}{2n}\left(\|\boldsymbol{\omega}\|^2 + \gamma^2\right) + C_N \sum_{j \in D_i} h(b_j(\mathbf{A}_j^\top \boldsymbol{\omega} + \gamma))$, where $D_i \subset \{1, 2, \ldots, N\}$ is an index set for data points of agent i and N is the total number of data points. We choose the size of the data set for each local function to be a constant ($|D_i| = 50$), thus $N = 50n$. It is easy to check that each f_i has Lipschitz gradients and is strongly convex with $\mu_i = 1/n$.

We will compare our results with a centralized gradient descent algorithm, which updates every Γ_u iterations using the step-size sequence $\alpha_c(k) = \Gamma_u/(\mu k)$, in the direction of the *sum* of the gradients of all agents.

To make gradient estimates stochastic, we add a uniformly distributed noise $\boldsymbol{\varepsilon}_i \sim \mathbb{U}[-b/2, b/2]^d$ to the gradient estimates of each agent and $\boldsymbol{\varepsilon}_c \sim \mathbb{U}[-\sqrt{n}b/2, \sqrt{n}b/2]^d$ to the gradient of the centralized gradient descent, where $\mathbb{U}[b_1, b_2]^d$ denotes the uniform distribution of size d over the interval $[b_1, b_2)$, $b_1 < b_2$. Note that $\boldsymbol{\varepsilon}_i$ and $\boldsymbol{\varepsilon}_c$ are bounded and have zero mean and $\mathbb{E}[\|\boldsymbol{\varepsilon}_i\|^2] = db^2/12$ and $\mathbb{E}[\|\boldsymbol{\varepsilon}_c\|^2] = ndb^2/12$. We set $b = 4$ for all simulations.

Agents wake up with probability P_w and links fail with probability P_f, unless they reach their maximum allowed value where the algorithm forces the agent to wake up or the link to work successfully. The link delays are chosen uniformly between 1 to Γ_{del}.

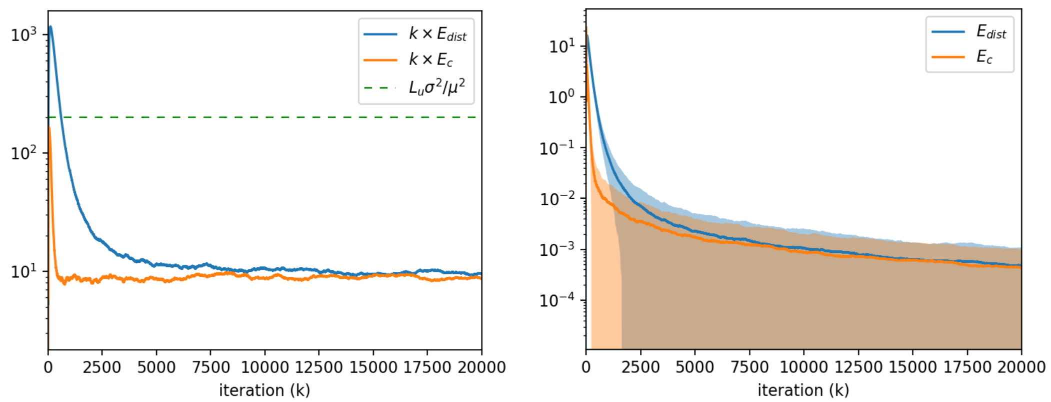
Each data set D_i is synthetically generated by picking 25 data points around each of the centers $(1, 1)$ and $(3, 3)$ with multivariate normal distributions, labeled -1 and $+1$, respectively. In generating strongly connected random graphs, we pick each edge with a probability of 0.5 and then check if the resulting graph is strongly connected; if it isn't, we repeat the process. Since the initial step-sizes for the distributed algorithm can be very large (e.g., $\alpha(1) = 50$ for $n = 50$), to stabilize the algorithms, both algorithms are started with $k_0 = 100$. This wouldn't affect the asymptotic convergence performance. Moreover, the initial point of the centralized algorithm and all agents in RASGP are chosen as $\mathbf{1}_d$.

Let us denote by $\hat{\mathbf{z}}(k) := (1/n)\sum_{i=1}^n \mathbf{z}_i(k)$ the average of $\mathbf{z}$-values of non-virtual agents. Then, we define *optimization errors* $E_{dist} := \|\hat{\mathbf{z}}(k) - \mathbf{z}^*\|^2$ and $E_c(k) := \|\mathbf{x}_c(k) - \mathbf{z}^*\|^2$ for RASGP and Centralized stochastic gradient descent, respectively.

Since our performance guarantees are for the expectation of (squared) errors, for each network setting, we perform up to 1000 Monte-Carlo simulations and use their corresponding performance to estimate the average behavior of the algorithms. Since accurately estimating the *true* expected value requires an extremely large number of simulations, in order to alleviate the effect of spikes and high variance, we take the following steps. First a batch of simulations are performed and their average is calculated. Next, to obtain a smoother plot, an average over every 100 iterations is taken. And finally, the median of these outputs over all the batches is our estimate of the expected value.

We report two figures for each setting: one including the errors E_{dist} and E_c, and another one including $k \times E_{dist}$ and $k \times E_c$ to demonstrate the convergence rates.

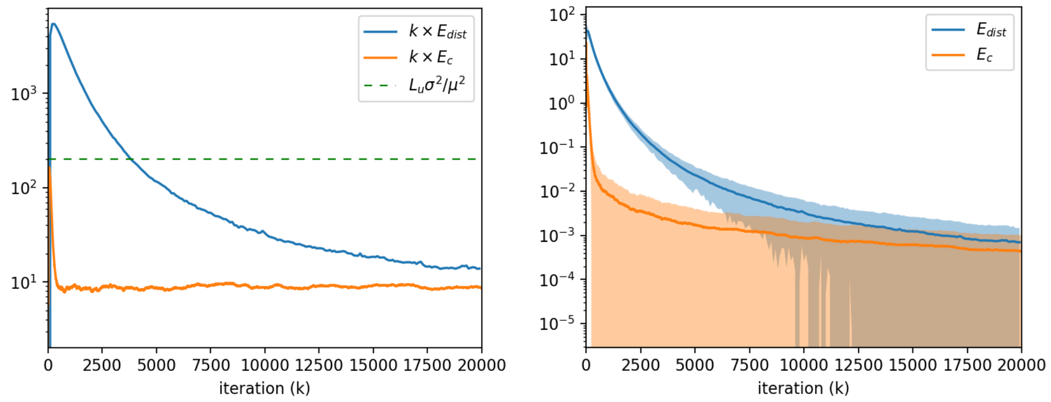
Finally, to study the non-asymptotic behavior of RASGP and its dependence on network size n, we have compared the performance of the centralized stochastic gradient descent



(a) k times squared errors.

(b) Errors and 1-standard-deviation band.

Figure 3: Results on a directed cycle graph of size $n = 50$, synchronous with no delays and link failures ($P_w = 1$, $P_f = 0$, $\Gamma_{\text{del}} = \Gamma_f = 0, \Gamma_u = 1$, $\Gamma_s = 2$).



(a) k times squared errors.

(b) Errors and 1-standard-deviation band.

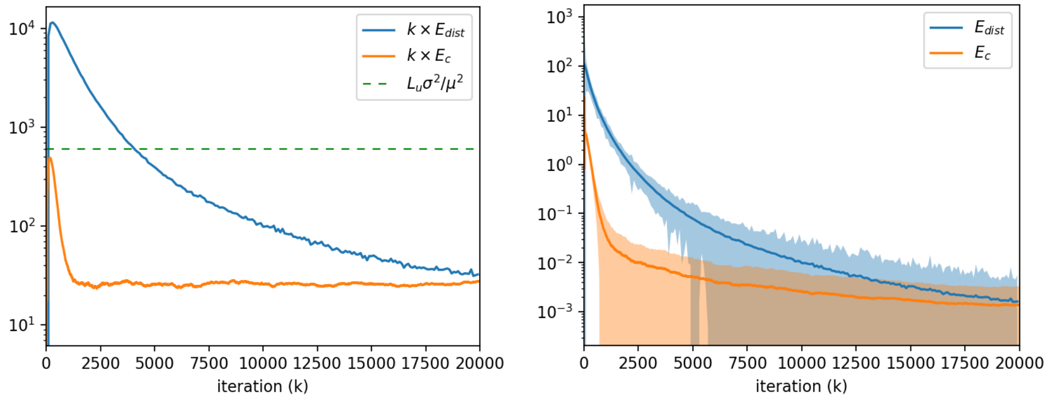
Figure 4: Results on a directed cycle graph of size $n = 50$, synchronous with delays and link failures ($P_w = 1$, $P_f = 0.3$, $\Gamma_{\text{del}} = \Gamma_f = 3, \Gamma_u = 1$, $\Gamma_s = 7$).

and RASGP over a bidirectional cycle graph, with error variances of $n^2\hat{\sigma}^2$ and $\sigma_i^2 = \hat{\sigma}^2$, respectively. Then, we plot the ratio $E_c(k)/E_{dist}(k)$ over n, for different iterations k.

4.2. Results

Our simulation results are consistent with our theoretical claims (due to the performance of centralized and decentralized methods growing closer over time) and show the achievement of an asymptotic network-independent convergence rate.

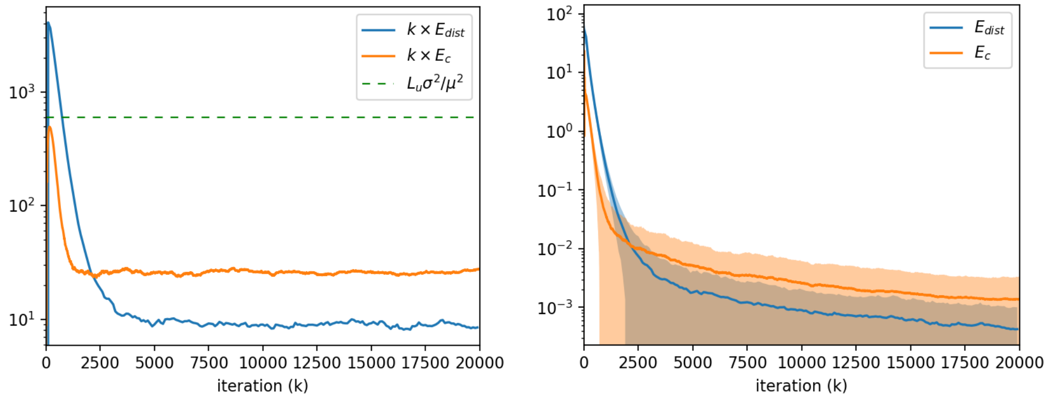
Fig. 3 shows that when there is no link failure or delay and all agents wake up at every iteration ($\Gamma_s = 2$), RASGP and centralized gradient descent have very similar performance. When we allow links to have delays and failures (see Fig. 4), as well as asynchronous updates (see Fig. 5), it takes longer for RASGP to reach its asymptotic convergence rate.



(a) k times squared errors.

(b) Errors and 1-standard-deviation band.

Figure 5: Results on a directed cycle graph of size $n = 50$, asynchronous with delays and link failures ($P_w = 0.5$, $P_f = 0.3$, $\Gamma_{\text{del}} = \Gamma_f = 3, \Gamma_u = 3$, $\Gamma_s = 17$).



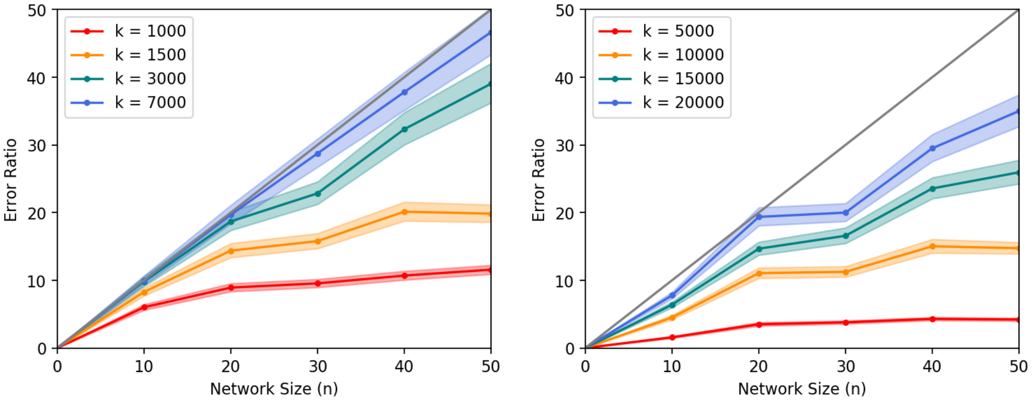
(a) k times squared errors.

(b) Errors and 1-standard-deviation band.

Figure 6: Results on a directed random graph of size $n = 50$, asynchronous with delays and link failures ($P_w = 0.5$, $P_f = 0.3$, $\Gamma_{\text{del}} = \Gamma_f = 3, \Gamma_u = 3$, $\Gamma_s = 17$).

We observe that, with all the other parameters fixed, the RASGP performs better on a random graph than on a cycle graph (see Figs. 5 and 6). A possible reason is that the cycle graph has a higher diameter or mixing time compared to the random graph, resulting in a slower decay of the consensus error.

We notice that by fixing the network size, increasing the number of iterations brings us closer to linear speed-up (see Fig. 7). On the other hand, when fixing the number of iterations, increasing the number of nodes, after a certain point, does not help speeding up the optimization. Moreover, by allowing link delays and failures (see Fig. 7b) we require more iterations to achieve network independence.



(a) Synchronous with no delays and link failures.

(b) Synchronous with delays and link failure ($P_w = 1$, $P_f = 0.3$, $\Gamma_{\text{del}} = \Gamma_f = 3, \Gamma_u = 1$, $\Gamma_s = 7$).

Figure 7: Error ratio over network size. Shaded areas correspond to 1-standard-deviation of the performance.

5. Conclusions

The main result of this paper is to stablish asymptotically, network independent performance for a distributed stochastic optimization method over directed graphs with message losses, delays, and asynchronous updates. Our work raises several open questions.

The most natural question raised by this paper concerns the size of the transients. How long must the nodes wait until the network-independent performance bound is achieved? The answer, of course, will depend on the network, but also on the number of nodes, the degree of asynchrony, and the delays. Understanding how this quantity scales is required before the algorithms presented in this work can be recommended to practitioners.

More generally, it is interesting to ask which problems in distributed optimization can achieve network-independent performance, even asymptotically. For example, the usual bounds for distributed subgradient descent (see, e.g., Nedic et al., 2018) depend on the spectral gap of the underlying network; various worst-case scalings with the number of nodes can be derived, and the final asymptotics are not network-independent. It is not immediately clear whether this is due to the analysis, or a fundamental limitation that will not be overcome.

Acknowledgments

The authors acknowledge support for this project by the AFOSR under grant FA9550-15-1-0394, by the ONR under grant N000014-16-1-224 and MURI N00014-19-1-2571, by the NSF under grants IIS-1914792, DMS-1664644, and CNS-1645681, and by the NIH under grant 1R01GM135930. A preliminary version of the results in Section 2 has been published in the proceedings of the *American Control Conference* 2018 (Olshevsky et al., 2018).

x

Appendix A. Proof of Lemma 4

Proof We use mathematical induction. For $k = 0$ we have $x^l_{ij}(0) = 0, \; \forall l$ and $u^x_{ij}(0) = \phi^x_i(0) = \rho^x_{ji}(0) = 0$. By (6) and the definition of u^x_{ij} and x^l_{ij} we obtain,

$$\begin{aligned} \rho^x_{ji}(1) &= 0, \\ u^x_{ij}(1) &= (1 - \sum_{l=1}^{\Gamma_d} \tau^l_{ij}(0))\phi^x_i(1), \\ \sum_{l=1}^{\Gamma_d} x^l_{ij}(1) &= (\sum_{l=1}^{\Gamma_d} \tau^l_{ij}(0))\phi^x_i(1). \end{aligned}$$

Equation (12) is concluded from first equation above and (13) results by summing up all three equations above.

Now assume this lemma is true for $k = 0, \ldots, K-1$. We want to show it will be true for $k = K$ as well. In the following, LHS and RHS denote the left-hand-side and right-hand-side of (12) for $k = K$. By (6) we have,

$$LHS = \sum_{l=1}^{\Gamma_d} \tau^l_{ij}(K-l)[\phi^x_i(K+1-l) - \rho^x_{ji}(K)].$$

Using (11) we obtain,

$$RHS = \sum_{l=1}^{\Gamma_d} \tau^l_{ij}(K-l)\upsilon^x_{ij}(K-l).$$

Hence, it suffices to show that:

$$\sum_{l=1}^{\Gamma_d} \tau^l_{ij}(K-l)[\phi^x_i(K+1-l) - \rho^x_{ji}(K) - \upsilon^x_{ij}(K-l)] = 0. \tag{50}$$

By part (e) of Assumption 1, at most one of the $\tau^l_{ij}(K-l)$, $l = 1, \ldots, \Gamma_d$ is non-zero. If all are zeros, the result follows. Now suppose $\tau^l_{ij}(K-l) = 1$ for some l. Equation (50) becomes,

$$\phi^x_i(K+1-l) - \rho^x_{ji}(K) - \upsilon^x_{ij}(K-l) = 0.$$

Plugging in the definition of υ^x_{ij}, after rearrangement we obtain,

$$\phi^x_i(K-l) - u^x_{ij}(K-l) = \rho^x_{ji}(K). \tag{51}$$

By the induction hypothesis, (12) holds for $k = K-t$, $t = 1, \ldots, l$. Therefore,

$$\rho^x_{ji}(K+1-t) - \rho^x_{ji}(K-t) = x^1_{ij}(K-t).$$

Hence,

$$\begin{aligned}
\rho^x_{ji}(K) &= \rho^x_{ji}(K-l) + \sum_{t=1}^{l}(\rho^x_{ji}(K+1-t) - \rho^x_{ji}(K-t)) \\
&= \rho^x_{ji}(K-l) + \sum_{t=1}^{l} x^1_{ij}(K-t) \\
&= \rho^x_{ji}(K-l) + \sum_{l'=1}^{l} x^{l'}_{ij}(K-l) \qquad \text{(Lemma 3)} \\
&= \rho^x_{ji}(K-l) + \sum_{l'=1}^{d} x^{l'}_{ij}(K-l). \qquad \text{(Lemma 2)}
\end{aligned}$$

Moreover, by the induction hypothesis, (13) holds for $k = K - l$, thus,

$$\phi^x_i(K-l) - u^x_{ij}(K-l) = \rho^x_{ji}(K-l) + \sum_{l'=1}^{\Gamma_d} x^{l'}_{ij}(K-l).$$

Combining the two relations above we conclude (51).

To show (13), consider the following equations which are direct results of the definitions and (12) that we just showed for $k = K$:

$$\begin{aligned}
u^x_{ij}(K+1) &= (1 - \sum_{l=1}^{\Gamma_d} \tau^l_{ij}(K))\upsilon^x_{ij}(K), \\
\rho^x_{ji}(K+1) &= \rho^x_{ji}(K) + x^1_{ij}(K), \\
\sum_{l=1}^{\Gamma_d} x^l_{ij}(K+1) &= \sum_{l=2}^{\Gamma_d} x^l_{ij}(K) + \sum_{l=1}^{\Gamma_d} \tau^l_{ij}(K)\upsilon^x_{ij}(K).
\end{aligned}$$

Summing up both sides of the equations above we have,

$$\begin{aligned}
LHS &= u^x_{ij}(K+1) + \rho^x_{ji}(K+1) + \sum_{l=1}^{\Gamma_d} x^l_{ij}(K+1), \\
RHS &= \sum_{l=1}^{\Gamma_d} x^l_{ij}(K) + \rho^x_{ji}(K) + \upsilon^x_{ij}(K) \\
&= \sum_{l=1}^{\Gamma_d} x^l_{ij}(K) + \rho^x_{ji}(K) + u^x_{ij}(K) - \phi^x_i(K) + \phi^x_i(K+1) = \phi^x_i(K+1).
\end{aligned}$$

The last equality holds because of the induction hypothesis (13) for $k = K - 1$, hence completing the proof. ■

References

Alekh Agarwal and John C Duchi. Distributed delayed stochastic optimization. In *Advances in Neural Information Processing Systems*, pages 873–881, 2011.

Mohammad Akbari, Bahman Gharesifard, and Tamás Linder. Distributed online convex optimization on time-varying directed graphs. *IEEE Transactions on Control of Network Systems*, 4(3):417–428, 2017.

Tansu Alpcan and Christian Bauckhage. A distributed machine learning framework. In *48h IEEE Conference on Decision and Control (CDC) held jointly with 2009 28th Chinese Control Conference*, pages 2546–2551. IEEE, 2009.

Mahmoud Assran and Michael Rabbat. Asynchronous subgradient-push. *arXiv preprint arXiv:1803.08950*, 2018.

Florence Bénézit, Vincent Blondel, Patrick Thiran, John Tsitsiklis, and Martin Vetterli. Weighted gossip: Distributed averaging using non-doubly stochastic matrices. In *2010 IEEE International Symposium on Information Theory (ISIT)*, pages 1753–1757. IEEE, 2010.

Theodora S Brisimi, Ruidi Chen, Theofanie Mela, Alex Olshevsky, Ioannis Ch Paschalidis, and Wei Shi. Federated learning of predictive models from federated electronic health records. *International journal of medical informatics*, 112:59–67, 2018.

Tsung-Hui Chang, Mingyi Hong, Wei-Cheng Liao, and Xiangfeng Wang. Asynchronous distributed ADMM for large-scale optimizationpart I: algorithm and convergence analysis. *IEEE Transactions on Signal Processing*, 64(12):3118–3130, 2016a.

Tsung-Hui Chang, Wei-Cheng Liao, Mingyi Hong, and Xiangfeng Wang. Asynchronous distributed ADMM for large-scale optimizationpart II: Linear convergence analysis and numerical performance. *IEEE Transactions on Signal Processing*, 64(12):3131–3144, 2016b.

Jianshu Chen and Ali H Sayed. On the learning behavior of adaptive networkspart II: Performance analysis. *IEEE Transactions on Information Theory*, 61(6):3518–3548, 2015.

Paolo Di Lorenzo and Gesualdo Scutari. Next: In-network nonconvex optimization. *IEEE Transactions on Signal and Information Processing over Networks*, 2(2):120–136, 2016.

Alejandro D Dominguez-Garcia and Christoforos N Hadjicostis. Distributed matrix scaling and application to average consensus in directed graphs. *IEEE Transactions on Automatic Control*, 58(3):667–681, 2013.

Alejandro D Domínguez-García and Christoforos N Hadjicostis. Convergence rate of a distributed algorithm for matrix scaling to doubly stochastic form. In *53rd IEEE Conference on Decision and Control*, pages 3240–3245. IEEE, 2014.

Hamid Reza Feyzmahdavian, Arda Aytekin, and Mikael Johansson. An asynchronous minibatch algorithm for regularized stochastic optimization. *IEEE Transactions on Automatic Control*, 61(12):3740–3754, 2016.

Bahman Gharesifard and Jorge Cortés. Distributed strategies for generating weight-balanced and doubly stochastic digraphs. *European Journal of Control*, 18(6):539–557, 2012.

Christoforos N Hadjicostis, Nitin H Vaidya, and Alejandro D Domínguez-García. Robust distributed average consensus via exchange of running sums. *IEEE Transactions on Automatic Control*, 61(6):1492–1507, 2016.

Christoforos N Hadjicostis, Alejandro D Domínguez-García, Themistokis Charalambous, et al. Distributed averaging and balancing in network systems: with applications to coordination and control. *Foundations and Trends® in Systems and Control*, 5(2-3): 99–292, 2018.

Shibo He, Dong-Hoon Shin, Junshan Zhang, Jiming Chen, and Youxian Sun. Full-view area coverage in camera sensor networks: Dimension reduction and near-optimal solutions. *IEEE Transactions on Vehicular Technology*, 65(9):7448–7461, 2015.

Mingyi Hong. A distributed, asynchronous and incremental algorithm for nonconvex optimization: An ADMM approach. *IEEE Transactions on Control of Network Systems*, 2017.

David Kempe, Alin Dobra, and Johannes Gehrke. Gossip-based computation of aggregate information. In *Foundations of Computer Science, 2003. 44th Annual IEEE Symposium on*, pages 482–491. IEEE, 2003.

Anastasiia Koloskova, Sebastian Urban Stich, and Martin Jaggi. Decentralized stochastic optimization and gossip algorithms with compressed communication. *Machine Learning Research*, 97(CONF), 2019.

Guanghui Lan, Soomin Lee, and Yi Zhou. Communication-efficient algorithms for decentralized and stochastic optimization. *Mathematical Programming*, pages 1–48, 2018.

Mu Li, David G Andersen, Jun Woo Park, Alexander J Smola, Amr Ahmed, Vanja Josifovski, James Long, Eugene J Shekita, and Bor-Yiing Su. Scaling distributed machine learning with the parameter server. In *11th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 14)*, pages 583–598, 2014.

Xiangru Lian, Yijun Huang, Yuncheng Li, and Ji Liu. Asynchronous parallel stochastic gradient for nonconvex optimization. In *Advances in Neural Information Processing Systems*, pages 2737–2745, 2015.

Xiangru Lian, Ce Zhang, Huan Zhang, Cho-Jui Hsieh, Wei Zhang, and Ji Liu. Can decentralized algorithms outperform centralized algorithms? a case study for decentralized parallel stochastic gradient descent. In *Advances in Neural Information Processing Systems*, pages 5330–5340, 2017.

Xiangru Lian, Wei Zhang, Ce Zhang, and Ji Liu. Asynchronous decentralized parallel stochastic gradient descent. In *International Conference on Machine Learning (ICML)*, pages 3043–3052, 2018.

Ilan Lobel and Asuman Ozdaglar. Distributed subgradient methods for convex optimization over random networks. *IEEE Transactions on Automatic Control*, 56(6):1291–1306, 2010.

Fatemeh Mansoori and Ermin Wei. Superlinearly convergent asynchronous distributed network newton method. In *2017 IEEE 56th Annual Conference on Decision and Control (CDC)*, pages 2874–2879. IEEE, 2017.

Gemma Morral, Pascal Bianchi, Gersende Fort, and Jérémie Jakubowicz. Distributed stochastic approximation: The price of non-double stochasticity. In *Signals, Systems and Computers (ASILOMAR), 2012 Conference Record of the Forty Sixth Asilomar Conference on*, pages 1473–1477. IEEE, 2012.

Gemma Morral, Pascal Bianchi, and Gersende Fort. Success and failure of adaptation-diffusion algorithms with decaying step size in multiagent networks. *IEEE Transactions on Signal Processing*, 65(11):2798–2813, 2017.

Angelia Nedic. Asynchronous broadcast-based convex optimization over a network. *IEEE Transactions on Automatic Control*, 56(6):1337–1351, 2011.

Angelia Nedic and Alex Olshevsky. Distributed optimization over time-varying directed graphs. *IEEE Transactions on Automatic Control*, 60(3):601–615, 2015.

Angelia Nedic and Alex Olshevsky. Stochastic gradient-push for strongly convex functions on time-varying directed graphs. *IEEE Transactions on Automatic Control*, 61(12):3936–3947, 2016.

Angelia Nedic and Asuman Ozdaglar. Distributed subgradient methods for multi-agent optimization. *IEEE Transactions on Automatic Control*, 54(1):48–61, 2009.

Angelia Nedic, Alex Olshevsky, and Wei Shi. Achieving geometric convergence for distributed optimization over time-varying graphs. *SIAM Journal on Optimization*, 27(4): 2597–2633, 2017.

Angelia Nedic, Alex Olshevsky, and Michael G Rabbat. Network topology and communication-computation tradeoffs in decentralized optimization. *IEEE*, 106(5):953–976, 2018.

Arkadi Nemirovski, Anatoli Juditsky, Guanghui Lan, and Alexander Shapiro. Robust stochastic approximation approach to stochastic programming. *SIAM Journal on optimization*, 19(4):1574–1609, 2009.

Alex Olshevsky. Linear time average consensus and distributed optimization on fixed graphs. *SIAM Journal on Control and Optimization*, 55(6):3990–4014, 2017.

Alex Olshevsky, Ioannis Ch Paschalidis, and Artin Spiridonoff. Fully asynchronous push-sum with growing intercommunication intervals. *American Control Conference*, pages 591–596, 2018.

Boris N Oreshkin, Mark J Coates, and Michael G Rabbat. Optimization and analysis of distributed averaging with short node memory. *IEEE Transactions on Signal Processing*, 58(5):2850–2865, 2010.

Zhouhua Peng, Jun Wang, and Dan Wang. Distributed maneuvering of autonomous surface vehicles based on neurodynamic optimization and fuzzy approximation. *IEEE Transactions on Control Systems Technology*, 26(3):1083–1090, 2017.

Shi Pu and Alfredo Garcia. A flocking-based approach for distributed stochastic optimization. *Operations Research*, 66(1):267–281, 2017.

Shi Pu and Angelia Nedic. A distributed stochastic gradient tracking method. In *2018 IEEE Conference on Decision and Control (CDC)*, pages 963–968. IEEE, 2018.

Guannan Qu and Na Li. Harnessing smoothness to accelerate distributed optimization. *IEEE Transactions on Control of Network Systems*, 2017.

Guannan Qu and Na Li. Accelerated distributed Nesterov gradient descent. *IEEE Transactions on Automatic Control*, 2019.

Alexander Rakhlin, Ohad Shamir, Karthik Sridharan, et al. Making gradient descent optimal for strongly convex stochastic optimization. In *29th International Conference on Machine Learning (ICML)*, volume 12, pages 1571–1578. Citeseer, 2012.

S Sundhar Ram, Angelia Nedic, and Venugopal V Veeravalli. Distributed stochastic subgradient projection algorithms for convex optimization. *Journal of optimization theory and applications*, 147(3):516–545, 2010.

Benjamin Recht, Christopher Re, Stephen Wright, and Feng Niu. Hogwild: A lock-free approach to parallelizing stochastic gradient descent. In *Advances in Neural Information Processing Systems*, pages 693–701, 2011.

Jason DM Rennie and Nathan Srebro. Loss functions for preference levels: Regression with discrete ordered labels. In *IJCAI multidisciplinary workshop on advances in preference handling*, pages 180–186. Kluwer Norwell, MA, 2005.

Kevin Scaman, Francis Bach, Sébastien Bubeck, Yin Tat Lee, and Laurent Massoulié. Optimal algorithms for smooth and strongly convex distributed optimization in networks. In *34th International Conference on Machine Learning (ICML)-Volume 70*, pages 3027–3036. JMLR. org, 2017.

Wei Shi, Qing Ling, Gang Wu, and Wotao Yin. Extra: An exact first-order algorithm for decentralized consensus optimization. *SIAM Journal on Optimization*, 25(2):944–966, 2015.

Benjamin Sirb and Xiaojing Ye. Consensus optimization with delayed and stochastic gradients on decentralized networks. In *2016 IEEE International Conference on Big Data (Big Data)*, pages 76–85. IEEE, 2016.

Kunal Srivastava and Angelia Nedic. Distributed asynchronous constrained stochastic optimization. *IEEE Journal of Selected Topics in Signal Processing*, 5(4):772–790, 2011.

Lili Su and Nitin H Vaidya. Fault-tolerant multi-agent optimization: optimal iterative distributed algorithms. In *Proceedings of the 2016 ACM Symposium on Principles of Distributed Computing*, pages 425–434. ACM, 2016a.

Lili Su and Nitin H Vaidya. Non-bayesian learning in the presence of byzantine agents. In *International Symposium on Distributed Computing*, pages 414–427. Springer, 2016b.

Lili Su and Nitin H. Vaidya. Reaching approximate byzantine consensus with multi-hop communication. *Information and Computation*, 255:352 – 368, 2017. ISSN 0890-5401. doi: https://doi.org/10.1016/j.ic.2016.12.003. URL `http://www.sciencedirect.com/science/article/pii/S0890540116301262`. SSS 2015.

Ying Sun, Gesualdo Scutari, and Daniel Palomar. Distributed nonconvex multiagent optimization over time-varying networks. In *Signals, Systems and Computers, 2016 50th Asilomar Conference on*, pages 788–794. IEEE, 2016.

Ye Tian, Ying Sun, and Gesualdo Scutari. Asy-sonata: Achieving linear convergence in distributed asynchronous multiagent optimization. In *2018 56th Annual Allerton Conference on Communication, Control, and Computing (Allerton)*, pages 543–551. IEEE, 2018.

Konstantinos I Tsianos, Sean Lawlor, and Michael G Rabbat. Consensus-based distributed optimization: Practical issues and applications in large-scale machine learning. In *Communication, Control, and Computing (Allerton), 2012 50th Annual Allerton Conference on*, pages 1543–1550. IEEE, 2012a.

Konstantinos I Tsianos, Sean Lawlor, and Michael G Rabbat. Push-sum distributed dual averaging for convex optimization. In *2012 51st IEEE Conference on Decision and Control (CDC)*, pages 5453–5458. IEEE, 2012b.

John Tsitsiklis, Dimitri Bertsekas, and Michael Athans. Distributed asynchronous deterministic and stochastic gradient optimization algorithms. *IEEE Transactions on Automatic Control*, 31(9):803–812, 1986.

Tianyu Wu, Kun Yuan, Qing Ling, Wotao Yin, and Ali H Sayed. Decentralized consensus optimization with asynchrony and delays. *IEEE Transactions on Signal and Information Processing over Networks*, 4(2):293–307, 2018.

Chenguang Xi and Usman A Khan. Dextra: A fast algorithm for optimization over directed graphs. *IEEE Transactions on Automatic Control*, 62(10):4980–4993, 2017a.

Chenguang Xi and Usman A Khan. Distributed subgradient projection algorithm over directed graphs. *IEEE Transactions on Automatic Control*, 62(8):3986–3992, 2017b.

Chenguang Xi, Ran Xin, and Usman A Khan. Add-opt: Accelerated distributed directed optimization. *IEEE Transactions on Automatic Control*, 63(5):1329–1339, 2018.

Jinming Xu, Shanying Zhu, Yeng Chai Soh, and Lihua Xie. Augmented distributed gradient methods for multi-agent optimization under uncoordinated constant stepsizes. In *2015 54th IEEE Conference on Decision and Control (CDC)*, pages 2055–2060. IEEE, 2015.

Kun Yuan, Qing Ling, and Wotao Yin. On the convergence of decentralized gradient descent. *SIAM Journal on Optimization*, 26(3):1835–1854, 2016.