

A Distributed Laplacian Solver and its Applications to Electrical Flow and Random Spanning Tree Computation

Iqra Altaf Gillani¹ and Amitabha Bagchi¹

¹{iqraaltaf,bagchi}@cse.iitd.ac.in, Department of Computer Science and Engineering, IIT Delhi

Abstract

We present a distributed solver for a large and important class of Laplacian systems that we call “one-sink” Laplacian systems. Specifically, our solver can produce solutions for systems of the form $L\mathbf{x} = \mathbf{b}$ where exactly one of the coordinates of \mathbf{b} is negative. Our solver is an organically distributed algorithm that takes $\tilde{O}(t_{\text{hit}})$ rounds to produce an approximate solution where t_{hit} is the hitting time of the random walk on the graph, which is $\Theta(n)$ for a large set of important graphs. The \tilde{O} notation hides a dependency on error parameters and a logarithmic dependency on the inverse of λ_2^L , the second smallest eigenvalue of L . This constitutes an improvement over the past work on distributed solvers which have a linear dependence on $1/\lambda_2^L$. The class of one-sink Laplacians includes the important voltage computation problem and allows us to compute the effective resistance between nodes in a distributed setting assuming the CONGEST model, i.e., each message is $\Theta(\log n)$ in size. As a result, our Laplacian solver can be used to adapt the approach by Kelner and Mądry (2009) to give the first distributed algorithm to compute approximate random spanning trees efficiently.

Our solver, which we call “Distributed Random Walk-based Laplacian Solver” (DRW-LSolve) works by quickly approximating the stationary distribution of a multi-dimensional Markov chain induced by a queueing network model that we call the “data collection” process. In this process each node v for which $\mathbf{b}_v > 0$ generates data packets with a rate proportional to \mathbf{b}_v and the node for which $\mathbf{b}_v < 0$ acts as a sink. We show that when this multidimensional chain is ergodic the vector whose v th coordinate is proportional to the probability at stationarity of the queue at v being non-empty is a solution to $L\mathbf{x} = \mathbf{b}$. We estimate this solution in a natural way by statistically determining the proportion of time slots for which the queues are empty. A fast-mixing property of the multi-dimensional chain allows us to achieve the running time result.

Keywords: Laplacian solver, distributed algorithms, electrical flow, random spanning trees

1 Introduction

A number of fields including computer science, operations research, electrical engineering, machine learning, and computational biology, present important algorithmic problems that can be approached by solving a Laplacian system of equations. A pioneering paper of Spielman and Teng’s [26] proposed a quasi-linear time algorithm for solving Laplacian systems in 2004, and since then Laplacian solvers have been able to improve longstanding bounds for a wide range of fundamental graph theoretic problems, c.f., surveys by Spielman [25] and Vishnoi [27]. Moreover, if we view the graph as an electrical network with resistances on edges, the effective resistance between two nodes can be easily derived by solving a Laplacian system. As a result, a series of exciting graph algorithmic

developments that use electrical flow computation as a primitive also rely on Laplacian solvers, e.g., fast computation of max flows [9][18], generation of random spanning trees [13][19], and graph sparsification [25].

However, although the Laplacian systems are defined over a graph, and have been used to address problems that can arise naturally in distributed settings, there has been very little work on distributed Laplacian solvers. In this paper, we endeavor to fill that gap by describing an organically distributed algorithm that works in the well-studied CONGEST [20] model to solve a large and useful class of Laplacian systems that include electrical flow.

Several classes of centralized Laplacian solvers are known. The simpler ones, e.g., Richardson iteration or Chebyshev polynomial-based methods require performing matrix multiplications iteratively. The more sophisticated Kaczmarz method and Conjugate Gradient method require computing projections, i.e., dot products, and occasional matrix multiplications. As such, it is natural to assume that a distributed Laplacian solver can be built by performing these primitive operations in a distributed way and, in fact, Zouzias and Freris [28] do precisely this, adopting the Kaczmarz iteration to a gossip setting and producing a solution in $\tilde{O}(m/\lambda_2^L)$ time where λ_2^L is the second-smallest eigenvalue of the Laplacian L . Our algorithm has three advantages over this approach: (a) our number of rounds does not have a linear dependence on $1/\lambda_2^L$ which can be $\Theta(n^2)$ for some classes of graphs like the path graph and the cycle graph, (b) our method does not require any vector operations at the nodes of the network, just simple counting and division, making it more suitable for networks with low-end nodes with very limited computation, e.g., sensor networks or IoT networks, and, perhaps most importantly, (c) our method is a completely new approach to Laplacian systems based on the theory of random walks, multi-dimensional Markov chains and queueing.

Specifically, we formulate a stochastic problem which captures the properties of Laplacian systems of the form $L\mathbf{x} = \mathbf{b}$ with a constraint that only one element in \mathbf{b} is negative. We call such systems “one-sink” Laplacian systems since the stochastic process can be viewed as a network in which some nodes are generating data, and there is a single sink which collects all the packets that it receives. The class of one-sink Laplacians contains the important electrical flow problem. We call our stochastic process the “data collection problem” and show that this process has an equivalence to the one-sink Laplacian system at stationarity, provided it is ergodic and has a stationary distribution. We show that in its stationary state, the data collection system naturally contains a solution to the Laplacian system. The technical challenges that arise in deriving an algorithm from this observation are: (1) We have to ensure that our data collection problem is ergodic, (2) that we can get close to the stationarity in reasonable time and (3) once we are close to the stationarity we can estimate the solution in good time. We show that there is parameter range where the data collection process is not just ergodic but geometrically ergodic, i.e., starting from any state the distance from stationarity reduces exponentially for an appropriately chosen exponent. The proof of geometric ergodicity of the complex multi-dimensional Markov chain described by the data collection process is an elegant coupling-based proof and is of independent interest since it is difficult, in general, to prove such results for multi-dimensional chains.

Our results. Our setting is as follows: we are given an undirected weighted graph $G = (V, E, w)$ with $|V| = n$ nodes, $|E| = m$ edges, $w : E \rightarrow \mathbb{R}_+$, with adjacency matrix A such that $A_{uv} = w_{uv}$ if $(u, v) \in E$ and 0 otherwise. Let $D \in \mathbb{R}^{|V| \times |V|}$ be the diagonal matrix of generalized degrees such that $D_{uu} = d_u$, where $d_u = \sum_{v: (u,v) \in E} w_{uv}$. Then, the corresponding Laplacian matrix is $L = D - A$. The natural random walk defined on G has transition matrix \mathcal{P} where $\mathcal{P}[u, v] = w_{uv}/d_u$ for $(u, v) \in E$ and 0 otherwise. If $(X_i)_{i \geq 0}$ is an instance of the random walk on G then the *hitting time*, t_{hit} , of this

random walk is defined by $\max_{u,v \in V} \mathbb{E}[t : X_0 = u, X_t = v, X_i \neq v \text{ for } 0 < i < t]$, i.e., the expected time taken for the random walk started at u to reach v maximized over all pairs of vertices. The main result of our paper is this:

Theorem 1 (Distributed Laplacian Solver). *Given a graph $G = (V, E, w)$ and $\mathbf{b} \in \mathbb{R}^{|V|}$ such that for $1 \leq i \leq n-1$, $\mathbf{b}_i \geq 0$, and $\mathbf{b}_n = -\sum_{i=1}^{n-1} \mathbf{b}_i$, the corresponding Laplacian system is*

$$\mathbf{x}^T L = \mathbf{b}^T, \quad (1)$$

For $\kappa \in (0, 1)$ and error parameters $\epsilon_1, \epsilon_2 \in (0, 1)$ such that $\epsilon_1 + \epsilon_2 < 1/2$, there is a distributed solver $DRW\text{-}LSolve(\kappa, w, \mathbf{b}, \epsilon_1, \epsilon_2)$ working in the *CONGEST* model that takes

$$O\left(\left(t_{\text{hit}} \log \epsilon_1^{-1} + \frac{\log n}{\kappa^2 \epsilon_2^2}\right) \log \frac{d_{\max}}{\lambda_2^L}\right)$$

rounds, to produce a vector $\hat{\mathbf{x}}$ that is an approximate solution to (1), where $d_{\max} = \max_{u \in V} d_u$ and t_{hit} is the hitting time of the natural random walk on G .

Further, $\hat{\mathbf{x}}$ has the properties that (i) $\hat{\mathbf{x}}_v$ is computed at node $v \in V$, (ii) $\hat{\mathbf{x}}_i > 0$, $1 \leq i \leq n-1$, $\hat{\mathbf{x}}_n = 0$, and, (iii) if \mathbf{x} is an exact solution to (1) such that $\mathbf{x}_i > 0$, $1 \leq i \leq n-1$, $\mathbf{x}_n = 0$, then

$$|\hat{\mathbf{x}}_i - \mathbf{x}_i| \leq (\epsilon_1 + \epsilon_2) \mathbf{x}_i,$$

whenever $\kappa < \mathbf{x}_u d_u$.

The leading term in our number of rounds is $\tilde{O}(t_{\text{hit}} \log(1/\lambda_2^L))$ and for extreme cases like the barbell graph or the lollipop graph $t_{\text{hit}} = \Theta(n^3)$ [2] and so this implies poorer performance than the method of Zouzias and Freris [28] which takes $\tilde{O}(m/\lambda_2^L)$ rounds. However, for several important classes of graphs like expanders, random geometric graphs, the complete graph, the hypercube, the d -dimensional grid, Erdős-Rényi graphs, t_{hit} is known to be $\Theta(n)$ [2][5] which makes our performance better in such cases. Also for some graphs like the path graph and the cycle graph which have $t_{\text{hit}} = \Theta(n^2)$ [2] it would appear we have a disadvantage but since $\lambda_2^L = O(1/n^2)$ [1] for such graphs our solver outperforms Zouzias and Freris's algorithm.

The algorithm mentioned in Theorem 1 can be directly applied to compute effective resistance $R_{u,v}$ between a pair of node u, v by considering u as the source node and v as the sink.

Corollary 1 (Effective Resistance Computation). *Given an undirected weighted graph $G = (V, E, w)$ with a positive weight function and $|V| = n$ nodes there is a distributed algorithm to compute the effective resistance $R_{u,v}$ between any pair of nodes u, v within an $(\epsilon_1 + \epsilon_2)$ error such that $\epsilon_1 + \epsilon_2 < 1/2$ in*

$$O\left(\left(t_{\text{hit}}(u, v) \log \epsilon_1^{-1} + \frac{d_{\max}^2 \log n}{d_{\min}^2 \epsilon_2^2}\right) \log \frac{d_{\max}}{\lambda_2^L}\right)$$

rounds where $t_{\text{hit}}(u, v)$ is the expected time taken by the random walk starting from node u to hit node v , d_{\max}, d_{\min} are the maximum and minimum generalized degrees of graph G respectively, and λ_2^L is the second smallest eigenvalue of the Laplacian matrix of graph.

This distributed way of finding effective resistance in turn helps to compute the electrical flows. Using such distributed electrical flow computation as a subroutine we describe an algorithm for distributed random spanning tree generation by adapting the centralized algorithm of Kelner and Mądry [13].

Theorem 2 (Random Spanning Tree Generation). *Given an undirected bounded-degree graph $G = (V, E)$ with $|V| = n$ and $|E| = m$ edges, our Distributed RST Generation Algorithm computes random spanning tree of G from a distribution that is $(1 + \epsilon)$ of the uniform distribution with a running time of $\tilde{O}\left(\frac{m}{\sqrt{n}} t_{hit} \log \epsilon^{-1}\right)$ rounds where t_{hit} is the worst-case hitting time of random walk on graph G .*

To the best of our knowledge, we present the first distributed algorithm for this problem. Note that a straightforward application of Zouzias and Freris’s algorithm would yield a time complexity of $\tilde{\Omega}\left(\frac{m^2}{\lambda_2^L \sqrt{n}}\right)$ rounds. As such the random spanning tree problem also illustrates the superiority of our method over previous work on an important application.

Paper organization. We first introduce the notion of “One-sink” Laplacian systems in Section 2.1 and then we discuss the equivalence with a stochastic process on a graph, the data collection process, in Section 2.2. In Section 3, we present a distributed solver to use this equivalence to solve one-sink Laplacian systems. In Section 4, we discuss a fundamental graph theoretic problem that can be solved by using our distributed solver as a subroutine: ϵ -random spanning tree generation. We briefly review the literature in Section 5 and finally conclude in Section 6 with some general remarks and some directions for future work.

2 “One-sink” Laplacian systems and an equivalent stochastic process

In this section, we first introduce the notion of “One-Sink” Laplacian systems. After that, we discuss a stochastic process: data collection on a graph and show how the steady-state equations of this setting exactly mimic a Laplacian system.

2.1 “One-Sink” Laplacian systems

The Laplacian L is a singular matrix with its null space being the subspace generated by the vector $\mathbf{1}$ which contains 1 in all its coordinates. Hence for any vector \mathbf{x} ,¹ the inner product $\langle L\mathbf{x}, \mathbf{1} \rangle = 0$. Therefore, when we consider equations of the form $L\mathbf{x} = \mathbf{b}$, we must have that $\langle \mathbf{b}, \mathbf{1} \rangle = 0$. Within this space we focus on the special case in which there is exactly one coordinate i such that $\mathbf{b}_i < 0$. In such a case, clearly $\mathbf{b}_i = -\sum_{j \neq i} \mathbf{b}_j$. For reasons that will be clear shortly we will call such vectors *one-sink* vectors and we will refer to the Laplacian systems of the form

$$L\mathbf{x} = \mathbf{b} \tag{2}$$

as *one-sink* Laplacian systems. Noting that since L is symmetric we can rewrite Eq. (2) as follows.

$$\mathbf{x}^T L = \mathbf{b}^T \tag{3}$$

The form in (3) will be useful to us since in Section 2.2 we will define a stochastic process on a network whose stationary state will give us a solution in this form. At that point the “one-sink” terminology will also be clarified.

¹In the following we use bold letters, e.g., \mathbf{x} for column vectors and denote row vectors as the transpose of column vectors, e.g., \mathbf{x}^T .

One-sink Laplacians and effective resistance. The importance of one-sink Laplacian's can be estimated by observing that the very widely used notion of the effective resistance falls within it. To see this, recall (or see, e.g., [27]) that if L^+ is the Penrose-Moore pseudo-inverse of L then for an edge $u, v \in V$, then effective resistance between u and v ,

$$R_{u,v} = (\mathbf{e}_u - \mathbf{e}_v)^T L^+ (\mathbf{e}_u - \mathbf{e}_v), \quad (4)$$

where \mathbf{e}_w is the vector with 1 at coordinate w and 0 elsewhere. Set $\mathbf{b} = \mathbf{e}_u - \mathbf{e}_v$, which is clearly a one-sink vector, and multiply both sides of (3) to obtain

$$\mathbf{x}^T = (\mathbf{e}_u - \mathbf{e}_v)^T L^+,$$

and observe that the RHS of (4) is now simply $\mathbf{x}_u - \mathbf{x}_v$. So the effective resistance can be easily obtained by solving a one-sink Laplacian system.

2.2 An equivalent stochastic process: Data Collection on a Graph

In an earlier work we had defined a stochastic process on a graph which has the remarkable property that at stationarity its steady state behaviour provides the solution of a one-sink Laplacian system [12]. We now explain this stochastic process and the equivalence. This is critically important since our distributed algorithm essentially involves ensuring this system has reached closed to stationarity and then estimating the steady state solution.

On a weighted undirected graph $G = (V, E, w)$ we define a discrete-time network “data collection” scenario where some “source” nodes generate data packets according to independent Bernoulli distributions. One node is designated as the *sink* that “collects” the data, and all nodes except the sink relay the data in a sink-oblivious manner, i.e., they simply pick a packet from their queue at random and send it to a neighbour chosen according to the one-step distribution of the natural random walk defined on G . In other words, if node u has a packet in its queue at time t it will forward it to v with probability $\mathcal{P}[u, v]$. We present a formal definition:

Definition 1 (Data collection process). *On a connected undirected weighted network $G = (V, E, w)$ we identify a distinguished sink node, u_s that passively collects data, and a set of data sources $V_s \subseteq V \setminus \{u_s\}$. Each node in $V \setminus \{u_s\}$ has a queue in which it can store data packets. We define a relative rate vector $\mathbf{J} \in \mathbb{R}^{|V \setminus \{u_s\}|}$: for a source $v \in V_s$, $\mathbf{J}_v > 0$, for all other nodes in $V \setminus \{u_s\}$, $\mathbf{J}_v = 0$, and $J_{u_s} = -\sum_{v \in V \setminus \{u_s\}} J_v$.*

Now, given a parameter $\beta > 0$ such that $\beta \mathbf{J}_v \leq 1$ for all $v \in V_s$, we define a discrete-time multidimensional Markov chain $\{Q_t^\beta\}_{t \geq 0}$ supported on $\mathbb{N} \cup \{0\}^{|V \setminus \{u_s\}|}$ that we call as the data collection process with parameter β . On this network:

- *For $t \geq 0, v \in V \setminus \{u_s\}$, $Q_t^\beta(v)$ denotes the size of the queue at node v at time t .*
- *At each $t \geq 0$, node $v \in V_s$ generates a new packet as an independent Bernoulli process with parameter $\beta \mathbf{J}_v$ and places it in its queue.*
- *At each $t \geq 0$ each node u in $V \setminus \{u_s\}$ picks one packet uniformly at random from its queue if its queue is non empty. It picks a neighbour v according to probability w_{uv}/d_u and transmits that packet to v .*
- *On transmission, the packet is removed from u 's queue. If $v \in V \setminus \{u_s\}$ the packet is placed in v 's queue. If v is u_s then the packet is sunk, i.e., it is removed from the system.*

Clearly if the rate controlling parameter is too high β , $\|Q_t^\beta\|$ will tend to infinity in the limit, so for this definition to be useful there needs to be a regime of β values wherein this process achieves its steady state. In [12] it was shown that such a regime does exist.

Lemma 1 (Gillani et. al. [12]). *For the data collection process there exists a $\beta^* > 0$ such that the multidimensional Markov chain $\{Q_t^\beta\}_{t \geq 0}$ is ergodic for all $\beta < \beta^*$ and for $\beta \geq \beta^*$ the chain is non-ergodic. Specifically, for $\beta < \beta^*$,*

- $\{Q_t^\beta\}_{t \geq 0}$ has a stationary distribution and
- $\lim_{t \rightarrow \infty} |Q_t^\beta(v)|$ is a finite constant depending on β for each $v \in V \setminus \{u_s\}$.

2.2.1 Establishing the equivalence to one-sink Laplacian systems.

We now show the equivalence of the evolution of this stochastic process with the one-sink Laplacian system of Eq. (3). Now, we can write the basic one step queue evolution equation for any node $u \in V$ as

$$\mathbb{E}[Q_{t+1}(u) | Q_t(u)] = Q_t(u) - \mathbf{1}_{\{Q_t(u) > 0\}} \sum_{v: v \sim u} \mathcal{P}[u, v] + \sum_{v: v \sim u} \mathcal{P}[v, u] \mathbf{1}_{\{Q_t(v) > 0\}} + A_t(u) \quad (5)$$

where the second and third term on the right-hand side of the above equation represents the transmissions sent to and received from the neighbors respectively and $A_t(u)$ is the number of packets generated at u , which is 0 if $u \notin V_s$ and is 1 with probability $\beta \mathbf{J}_v$ if $v \in V_s$. Now, taking expectations on both sides of Eq. (5) and let $\eta_u^t = P[Q_t(u) > 0]$ be the queue occupancy probability of node u and observing that $\mathbb{E}[A_t(u)] = \beta \mathbf{J}_u$, we have

$$\mathbb{E}[Q_{t+1}(u)] = \mathbb{E}[Q_t(u)] - \eta_u^t \sum_{v: v \sim u} \mathcal{P}[u, v] + \sum_{v: v \sim u} \mathcal{P}[v, u] \eta_v^t + \beta \mathbf{J}_u. \quad (6)$$

By Lemma 1 we know that for an appropriately chosen value of β the process has a steady state. In a steady state $\mathbb{E}[Q_t(u)]$ is a constant, so if we let $\boldsymbol{\eta}_u$ be the queue occupancy probability of node u at the stationarity, then we have the steady-state equation for the given node as

$$-\boldsymbol{\eta}_u \sum_{v: v \sim u} \mathcal{P}[u, v] + \sum_{v: v \sim u} \mathcal{P}[v, u] \boldsymbol{\eta}_v + \beta \mathbf{J}_u = 0. \quad (7)$$

Rewriting this in vector form we get

$$\boldsymbol{\eta}^T (I - \mathcal{P}) = \beta \mathbf{J}^T. \quad (8)$$

Since $\mathcal{P} = D^{-1}A$, this can be rewritten as

$$\mathbf{x}^T L = \beta \mathbf{J}^T \quad (9)$$

where $\mathbf{x}^T = \boldsymbol{\eta}^T D^{-1}$ is a row vector such that $\mathbf{x}_u = \boldsymbol{\eta}_u / d_u$ for all u where $\boldsymbol{\eta}_u$ is the steady-state queue occupancy probability. Comparing this with (3) we see that they are identical and so $\boldsymbol{\eta}^T D^{-1} / \beta$ is a solution for (3).

2.2.2 Mapping the stationary state of the data collection process to a canonical solution of the one-sink Laplacian system.

As discussed before, L is a singular matrix with the subspace generated by $\mathbf{1}$ being its null space. So we normally expect to find a solution \mathbf{y} to have the property that $\langle \mathbf{y}, \mathbf{1} \rangle = 0$, i.e., $\sum_{i=1}^n y_i = 0$. But clearly \mathbf{x} obtained as from the steady state of the data collection process has all coordinates non-negative, and so it is not in the canonical form.

However, we can write any other solution to (8) as $\hat{\boldsymbol{\eta}} = \boldsymbol{\eta} + z\nu$ where ν is the stationary distribution of the Markov Chain with transition matrix \mathcal{P} , i.e., $\nu = \nu\mathcal{P}$ and $z \in \mathbb{R}$ is any constant. This is because we know $z\nu(I - \mathcal{P}) = 0$, so we have, $(\boldsymbol{\eta} + z\nu)(I - \mathcal{P}) = \boldsymbol{\eta}(I - \mathcal{P})$. So, by choosing the appropriate constant offset we can get to any other solution of the given Laplacian equation. Similarly, the problem of $\beta\mathbf{J}_v \leq 1$ on the right hand side of Eq. (8) can be handled by scaling the obtained solution by the appropriate value.

In summary we can use the procedure shown in Figure 1 for deriving the canonical solution to $\mathbf{x}^T L = \mathbf{b}^T$ (Eq. (3)) by solving the data collection problem $\mathbf{x}^T L = \beta\mathbf{J}^T$ (Eq. (9)). We mark with an asterisk (*) those steps which are non-trivial.

1. Create the network $G = (V, E, w)$ corresponding to the Laplacian L .
2. Since $b_v < 0$ for a single v , make that v the sink for the data collection process. Now, given v is the chosen sink, set $\mathbf{J} = \frac{1}{\sum_{i \neq v} b_i} \mathbf{b}$.
3. (*) Find a value of β for which the data collection process is ergodic.
4. (*) Let the data collection process converge to stationarity.
5. Measure the vector $\boldsymbol{\eta}$ by computing the fraction of time steps each queue is occupied.
6. Compute z^* such that $\langle (\boldsymbol{\eta}D^{-1} + z^*\nu), \mathbf{1} \rangle = 0$, where ν is the stationary distribution of the natural random walk, i.e., $\nu_v = d_v / \sum_{u \in V} d_u$.
7. Return the solution $\sum_{i \neq v} b_i [(\boldsymbol{\eta}D^{-1} + z^*\nu)] / \beta$.

Figure 1: Steps for computing solution to the Laplacian equation using data collection process

Finding the appropriate β (Step 3) is non-trivial. Reaching stationarity is impossible (Step 4), so we have to figure out how close we need to be. The measurements made in Step 5 will also introduce error. In Section 3, we present an algorithm that shows how to deal with these challenges.

2.2.3 A rate lower bound for the data collection process

We will see in Section 3 that executing the program given in Figure 1 will take time that depends inversely on the value of β^* . For proving upper bounds on the time we will need the following lower bound on β^* :

Lemma 2. *If β^* is the value such that the multidimensional Markov chain $\{Q_t^\beta\}_{t \geq 0}$ associated with a data collection process on $G = (V, E, w)$ with sink u_s , source set V_s , and relative rate vector \mathbf{J} is*

ergodic for $\beta < \beta^*$ and non-ergodic for $\beta \geq \beta^*$ then

$$\beta^* \geq \frac{\lambda_2^L}{2d_{\max} \sum_{v \neq u_s} \mathbf{J}_v}$$

Proof of Lemma 2. For the given Laplacian steady-state equation $\mathbf{x}^T L = \beta \mathbf{J}^T$, let \mathbf{x} be *any* solution and \mathbf{y} be a canonical solution, i.e., the solution for which $\mathbf{y}^T \mathbf{1} = 0$, i.e., $\sum \mathbf{y}_i = 0$. So, there must be some $w \in \mathbb{R}$ such that $\mathbf{x} = \mathbf{y} + w\mathbf{1}$. Now, let $\boldsymbol{\psi}_i^T, 1 \leq i \leq n$ be the normalized (left) eigenvectors of L corresponding to the eigenvalues $0 = \lambda_1^L \leq \lambda_2^L \leq \dots \leq \lambda_n^L$ such that $\boldsymbol{\psi}_i^T \boldsymbol{\psi}_i = 1, 1 \leq i \leq n$. Moreover, note that $\boldsymbol{\psi}_1 = \mathbf{1}$, so, $\boldsymbol{\psi}_i^T \mathbf{1} = 0$ for $2 \leq i \leq n$. So, rewriting the steady-state equation in terms of these eigenvectors we have

$$\sum_{i=2}^n \lambda_i^L (\mathbf{y}^T \boldsymbol{\psi}_i) \boldsymbol{\psi}_i = \beta \mathbf{J}^T.$$

Taking norms on both sides we get

$$\lambda_2^L \left\| \sum_{i=2}^n (\mathbf{y}^T \boldsymbol{\psi}_i) \boldsymbol{\psi}_i \right\| \leq \left\| \sum_{i=2}^n \lambda_i^L (\mathbf{y}^T \boldsymbol{\psi}_i) \boldsymbol{\psi}_i \right\| = \|\beta \mathbf{J}^T\|. \quad (10)$$

Now, since we know $\mathbf{y}^T \mathbf{1}$ is 0 and the eigenvectors $\boldsymbol{\psi}_i, 2 \leq i \leq n$ span the subspace orthogonal to $\mathbf{1}$, therefore, $\left\| \sum_{i=2}^n (\mathbf{y}^T \boldsymbol{\psi}_i) \boldsymbol{\psi}_i \right\| = \|\mathbf{y}\|$. Putting this back in (10) we have

$$\lambda_2^L \|\mathbf{y}\| \leq \beta \|\mathbf{J}^T\|. \quad (11)$$

We now try to find a lower bound on $\|\mathbf{y}\|$ for the specific case where $\beta = \beta^*$, the maximum stable rate for the data collection process. Let \mathbf{x} be the solution produced by the data collection process at $\beta = \beta^*$. Since $\mathbf{x}^T D$ is a vector of queue occupancy probabilities in the data collection scenario, we know that

1. $\mathbf{x}_i \geq 0, 1 \leq i \leq n$.
2. $\min_{i=1}^n \mathbf{x}_i = 0$.
3. $\max_{i=1}^n d_i \mathbf{x}_i = 1$.

From Fact 2 of this list we can deduce that if $\mathbf{x} = \mathbf{y} + w\mathbf{1}$ then $w = -\min_{i=1}^n \mathbf{y}_i$. Now, putting this into Fact 3 of the list we get $\max_{i=1}^n d_i (\mathbf{y}_i - \min_{j=1}^n \mathbf{y}_j) = 1$. So, we get that

$$\max_{i=1}^n \mathbf{y}_i - \min_{j=1}^n \mathbf{y}_j \geq \frac{1}{d_{\max}}$$

where d_{\max} is the maximum generalized degree of graph. Now, consider $a > 0$ and $b < 0$ such that $a - b = \ell$, then we know $a^2 + b^2$ achieves minimum value at $\ell^2/2$. Using this we get that

$$\left\{ \max_{i=1}^n \mathbf{y}_i \right\}^2 + \left\{ \min_{j=1}^n \mathbf{y}_j \right\}^2 \geq \frac{1}{2d_{\max}^2}$$

This gives us the lower bound $\|\mathbf{y}\| \geq \frac{1}{\sqrt{2}d_{\max}}$. Putting this back in Eq. (11) we get $\beta^* \|\mathbf{J}\| \geq \frac{\lambda_2^L}{\sqrt{2}d_{\max}}$.

Further, note that $\|\mathbf{J}\|^2 = \sum_{v \neq u_s} \mathbf{J}_v^2 + \left(\sum_{v \neq u_s} \mathbf{J}_v \right)^2 \leq 2 \left(\sum_{v \neq u_s} \mathbf{J}_v \right)^2$. So, we get that

$$\beta^* \geq \frac{\lambda_2^L}{2d_{\max} \sum_{v \neq u_s} \mathbf{J}_v}. \quad (12)$$

□

3 Distributed Solver

3.1 The algorithm

We now present the main distributed algorithm. Our main solver algorithm, DRW-LSolve, is di-

Algorithm DRW-LSolve ($\kappa, w, \mathbf{b}, \epsilon_1, \epsilon_2$) Run by controller node u_s

```

1:  $T_{mix} \leftarrow 64t_{hit} \log \epsilon_1^{-1}, T_{samp} \leftarrow \frac{4 \log n}{\kappa^2 \epsilon_2^2}$ 
2: Send  $T_{mix}, T_{samp}, \sum_{v \in V} d_v, \sum_{v \neq u_s} \mathbf{b}_v$  to all  $u \in V \setminus \{u_s\}$ 
3:  $\mathbf{J} \leftarrow \mathbf{b} / \left( \sum_{v \neq u_s} \mathbf{b}_v \right)$ 
4: For every  $u \in V_s$  send  $\mathbf{J}_u$  to  $u$ 
5:  $\beta \leftarrow 1$  /* End of Initializations */
6: repeat
7:    $\beta \leftarrow \beta/2$  /* First value of  $\beta$  is  $1/2$  */
8:   Send message "Initiate DRW-Compute with rate  $\beta$ " to all nodes
9:   WAIT to receive  $\hat{\boldsymbol{\eta}}_u$  values from  $u \in V \setminus \{u_s\}$  and then compute  $\hat{\boldsymbol{\eta}}_{\max} \leftarrow \max_{u \in V \setminus \{u_s\}} \hat{\boldsymbol{\eta}}_u$ 
10: until  $\hat{\boldsymbol{\eta}}_{\max} < 3/4(1 - (\epsilon_1 + \epsilon_2))$  /* i.e.,  $\beta < 3\beta^*/4$  */
11: Compute  $z^* \leftarrow -\sum_{u \in V \setminus \{u_s\}} \hat{\boldsymbol{\eta}}_u / d_u$ 
12: Send message "DRW-LSolve over, compute output with shift factor  $z^*$ " to all nodes
13:  $\hat{\mathbf{x}}_{u_s} \leftarrow \frac{z^* d_{u_s}}{\sum_{v \in V} d_v} \frac{\sum_{v \neq u_s} \mathbf{b}_v}{\beta}$ 
14: return  $\hat{\mathbf{x}}_{u_s}$ 

```

rected by a single node, the sink u_s . We assume the value of t_{hit} has been precomputed and given to the controller. The central loop of the main algorithm extends from Line 6 to Line 10 where it works downwards from $\beta = 1/2$ to find a value of $\beta < \beta^*$. To do so it asks all the nodes to run the subroutine DRW-Compute with the current value of β . When the nodes have finished running DRW-Compute they have computed an approximate version of their occupancy probability for the data collection problem with parameter β and they send this back to the controller node, i.e., u_s . If the maximum occupancy probability received is below the threshold specified in Line 10 then the controller node is satisfied that the last value of β is at most $3\beta^*/4$ and it calls a halt to the algorithm. As mentioned in point 6 of Figure 1 we need to shift and scale $\hat{\boldsymbol{\eta}}$ to obtain a canonical solution, so the controller computes the shift factor z^* (Lines 11-12). Finally it computes its own value of the solution and returns it. All nodes apart from the controller, u_s , run DRW-LSolve-Slave.

Algorithm DRW-LSolve-Slave Run by nodes $u \in V \setminus \{u_s\}$

```

1: Receive parameters  $T_{mix} \leftarrow 64t_{hit} \log \epsilon_1^{-1}, T_{samp} \leftarrow \frac{4 \log n}{\kappa^2 \epsilon_2^2}, \sum_{v \in V} d_v, \sum_{v \neq u_s} \mathbf{b}_v, \mathbf{J}_u$  if  $u \in V_s$  from controller  $u_s$ 
2: if Message received "Initiate DRW-Compute with rate  $\beta$ " from  $u_s$  then
3:   Run DRW-Compute ( $\beta$ ) to compute  $\hat{\boldsymbol{\eta}}_u$ 
4:   Send  $\hat{\boldsymbol{\eta}}_u$  to controller node  $u_s$ 
5: else if Message received "DRW-LSolve over, compute output with shift factor  $z^*$ " then
6:    $\hat{\mathbf{x}}_u \leftarrow \left( \frac{\hat{\boldsymbol{\eta}}_u}{d_u} + \frac{z^* d_u}{\sum_{v \in V} d_v} \right) \frac{\sum_{v \neq u_s} \mathbf{b}_v}{\beta}$ 
7:   return  $\hat{\mathbf{x}}_u$ 
8: end if

```

They simply initiate DRW-Compute at the controller's direction with the appropriate value of β

and finally when the controller informs them that the algorithm is over they shift and scale their values of the occupancy probability and return the answer.

Algorithm DRW-Compute (β) Run by node $u \in V \setminus \{u_s\}$

Require: $T_{mix} \leftarrow 64t_{hit} \log \epsilon_1^{-1}$, $T_{samp} \leftarrow \frac{4 \log n}{\kappa^2 \epsilon_2^2}$, \mathbf{J}_u if $u \in V_s$, and w_{uv} for $v : u \sim v$

- 1: Initialize timer $T = 0$, $Q_t(u) = 0$, and $cnt = 0$
- 2: **repeat**
- 3: **if** $u \in V_s$ **then**
- 4: Generate a data packet with probability $\beta \mathbf{J}_u$ and place in queue.
- 5: **end if**
- 6: **if** $Q_t(u)$ is non-empty **then**
- 7: u picks a neighbor v with probability $\mathcal{P}[u, v] = \frac{w_{uv}}{\sum_{v: u \sim v} w_{uv}}$.
- 8: v adds packet p in $Q_{t+1}(v)$
- 9: u deletes packet p from $Q_{t+1}(u)$
- 10: **end if**
- 11: **if** $T \geq T_{mix}$ **then**
- 12: $cnt \leftarrow cnt + 1$
- 13: **end if**
- 14: $T \leftarrow T + 1$
- 15: **until** $T \leq T_{mix} + T_{samp}$
- 16: Send $\frac{cnt}{T_{samp}}$ to controller node u_s /* Queue occupancy probability estimate $\hat{\eta}_u = \frac{cnt}{T_{samp}}$ */

In the subroutine DRW-Compute each node simply simulates the data collection process with the given parameter and sends its estimate of its occupancy probability to the controller after the allotted time is over.

3.2 Analysis

DRW-LSolve works by repeatedly calling DRW-Compute till it finds a stable data rate $3\beta^*/8 \leq \beta < 3\beta^*/4$ and computes the solution to be output at that value of β . Clearly the correctness of DRW-LSolve depends on the ability of DRW-Compute to return a good approximation of the occupancy probabilities involved in the given number of rounds. So, we need the following lemmas that characterize the behaviour of DRW-Compute above and below β^* .

Lemma 3. *Given a data rate $\beta < \beta^*$, DRW-Compute (β) returns an estimate $\hat{\eta}_u$ for all $u \in V \setminus \{u_s\}$ such that $|\hat{\eta}_u - \eta_u| \leq (\epsilon_1 + \epsilon_2)\eta_u$ for $\kappa < \eta_u$ in time $\left(64t_{hit} \log \epsilon_1^{-1} + \frac{4 \log n}{\kappa^2 \epsilon_2^2}\right)$ where t_{hit} is the worst-case hitting time of random walk on the underlying graph.*

Lemma 3 is a mixing time result for the chain Q_t which is guaranteed to be ergodic when $\beta < \beta^*$. However, a key insight used by DRW-LSolve is that when $\beta > \beta^*$ there is bound to be one queue whose occupancy probability rises towards 1 if we wait long enough.

Lemma 4. *Given a data rate $\beta \geq \beta^*$, DRW-Compute (β) returns an estimate $\hat{\eta}_u$ for all $u \in V \setminus \{u_s\}$ such that $\hat{\eta}_{max} = \max_{u \in V \setminus \{u_s\}} \hat{\eta}_u \geq 1 - (\epsilon_1 + \epsilon_2)$ in time $\left(64t_{hit} \log \epsilon_1^{-1} + \frac{4 \log n}{\kappa^2 \epsilon_2^2}\right)$ where t_{hit} is the worst-case hitting time of random walk on the underlying graph.*

DRW-LSolve uses the insight of Lemma 4 to revise the value of β downwards by a factor of 2 if DRW-Compute finds a queue which has a very high occupancy. If, on the other hand, there is no

such queue, then Lemma 3 tells us that the occupancy probability vector is correctly approximated and we are done. Formally:

Lemma 5. *DRW-LSolve($\kappa, w, \mathbf{b}, \epsilon_1, \epsilon_2$) returns a $1 \pm (\epsilon_1 + \epsilon_2)$ -approximate solution to the Laplacian $\mathbf{x}^T L = \mathbf{b}^T$ in time*

$$\left(64t_{\text{hit}} \log \epsilon_1^{-1} + \frac{4 \log n}{\kappa^2 \epsilon_2^2} + 2\text{diam}(G)\right) \log \frac{16d_{\max}}{3\lambda_2^L}$$

where t_{hit} is the worst-case hitting time of random walk on the underlying graph, $\text{diam}(G)$ is the diameter of graph, d_{\max} is its generalized maximum degree, and λ_2^L is the second smallest eigenvalue of the Laplacian of graph.

Proof of Lemma 5. We will first prove the correctness of our solver DRW-LSolve based on the parameters set for the computation. After that we will bound the time for computing those correct estimates.

Correctness: As discussed before, our distributed solver works by choosing sink node u_s as a controller of algorithm whose job is to coordinate all other nodes and compute the stable data rate for operation. For this it starts a binary search from $\beta = 1/2$ and halves the data rate whenever it is found to be unstable. We operate at rate $3\beta^*/4$ such that we use Lemma 4 as a test condition for instability i.e., if $\beta \geq 3\beta^*/4$ then $\hat{\eta}_{\max} \geq 3/4(1 - (\epsilon_1 + \epsilon_2))$ where $\hat{\eta}_{\max}$ is computed using estimates $\hat{\eta}_u$ returned by all nodes $u \in V \setminus \{u_s\}$ using DRW-Compute subroutine. This rate of operation works fine for our solver as we need to correctly identify unstable data rates $\beta \geq \beta^*$ and for all such data rates, since these are greater than $3\beta^*/4$ as well, we are able to correctly identify them. However, for rates $3\beta^*/4 \leq \beta < \beta^*$, although these are stable our solver might indicate them as unstable. But, this is okay as we finally need a stable data rate for computation and by the definition of our binary search, the value at which it would stop i.e., $\hat{\eta}_{\max} < 3/4(1 - (\epsilon_1 + \epsilon_2))$ is at least $3\beta^*(1 - (\epsilon_1 + \epsilon_2))/8$ which is also a stable data rate. Moreover, from Lemma 3 we know for stable data rates DRW-Compute returns a $(1 - (\epsilon_1 + \epsilon_2))$ estimate of queue occupancy probability which can then be used to return the solution to the original Laplacian equation $\mathbf{x}^T L = \mathbf{b}^T$ after appropriate scaling.

Time: Now, let us bound the time taken by the solver to compute the estimates. We know that the solver performs binary search from $\beta = 1/2$ and halves the data rate each time it is found to be unstable. Also the least value it can reach is $3\beta^*(1 - (\epsilon_1 + \epsilon_2))/8$. Let ℓ be the number of such binary search iterations till it finds the stable data rate. In each such iteration, nodes $u \in V \setminus \{u_s\}$ in parallel use subroutine DRW-Compute and there is a message exchange (parameters from the controller to others and $\hat{\eta}_u$ values from nodes to the controller) which can take maximum upto $2\text{diam}(G)$ time where $\text{diam}(G)$ is the diameter of the graph. So from Lemma 3, the time taken for one iteration is $\left(64t_{\text{hit}} \log \epsilon_1^{-1} + \frac{4 \log n}{\kappa^2 \epsilon_2^2} + 2\text{diam}(G)\right)$. Since the binary search begins at $\beta = 1/2$ and ends above $3\beta^*(1 - (\epsilon_1 + \epsilon_2))/8$, we know that the number of iterations $\ell \leq \log(4/3\beta^*(1 - (\epsilon_1 + \epsilon_2)))$. If $\epsilon_1 + \epsilon_2 < 1/2$, then $\ell \leq \log(8/3\beta^*)$. Also, from Lemma 2 we know $\beta^* \geq \frac{\lambda_2^L}{2d_{\max}}$ as $\sum_{v \neq u_s} \mathbf{J}_v = 1$. So, the overall running time for DRW-LSolve($\kappa, w, \mathbf{b}, \epsilon_1, \epsilon_2$) is

$$\left(64t_{\text{hit}} \log \epsilon_1^{-1} + \frac{4 \log n}{\kappa^2 \epsilon_2^2} + 2\text{diam}(G)\right) \log \frac{16d_{\max}}{3\lambda_2^L}. \quad (13)$$

As the diameter of graph cannot be greater than the worst-case hitting time of random walk on it, $\text{diam}(G)$ term in the running time result can be suppressed. \square

The proofs of correctness of the subroutine DRW-Compute (Lemma 3 and Lemma 4) and the function DRW-LSolve (Lemma 5) together prove Theorem 1. So, we now turn to the analysis of DRW-Compute which is the heart of our method.

3.3 Analyzing DRW-Compute

3.3.1 Analyzing DRW-Compute when $\beta < \beta^*$

For the case of stable data rates $\beta < \beta^*$, from Lemma 1 we know that for such rates the Markov chain Q_t defined on the queue size of nodes is ergodic and has a stationary distribution. To estimate steady-state queue occupancy probabilities we need to be close to stationarity so we first bound the mixing time for Q_t .

Lemma 6. *Given an irreducible and aperiodic Markov chain Q_t described by the data collection process defined on $(\mathbb{N} \cup \{0\})^{|V|-1}$ having transition matrix $\mathcal{P}[\cdot, \cdot]$ and a stationary distribution π . For $\beta < \beta^*$ we have $\|\mathcal{P}^t(0, \cdot) - \pi\|_{TV} \leq \epsilon$ for $t = 64t_{hit} \log \epsilon^{-1}$, where t_{hit} is the worst-case hitting time of random walk on the underlying graph.*

Proof of Lemma 6. We first note that our Markov chain Q_t is *stochastically ordered* (c.f. [16]). To understand what this means we define a natural partial order on $\mathbb{N} \cup \{0\}^{|V|-1}$ as follows: $\mathbf{x} \preceq \mathbf{y}$ if $x_v \leq y_v$ for all $v \in V$. A function $f : \mathbb{N} \cup \{0\}^{|V|} \rightarrow \mathbb{R}$ is said to be increasing if $\mathbf{x} \preceq \mathbf{y}$ implies that $f(\mathbf{x}) \leq f(\mathbf{y})$. Given two random processes X and Y supported on $\mathbb{N} \cup \{0\}^{|V|-1}$ we say X is stochastically dominated by Y if $E[f(X)] \leq E[f(Y)]$ for every increasing function f . We now state the stochastic orderedness property as a claim.

Claim 1. *Given two instances of the data collection process Q_t and Q'_t such that $Q_0 \preceq Q'_0$, Q_t is stochastically dominated by Q'_t , $t \geq 0$. In particular this means that $P[Q_t(v) > 0] \leq P[Q'_t(v) > 0]$ for all $v \in V \setminus \{u_s\}$.*

The proof of this claim follows by constructing a coupling between the two chains such that each of them perform exactly the same transmission actions. In case one of the chains is empty then the transmission action is a dummy action. It is easy to see that stochastic ordering follows naturally for the data collection chain.

To use this claim, for our irreducible and aperiodic Markov chain Q_t described by the data collection process defined on $(\mathbb{N} \cup \{0\})^{|V|-1}$ having transition matrix $\mathcal{P}[\cdot, \cdot]$ and a stationary distribution π , let us define another irreducible and aperiodic Markov chain \bar{Q}_t with state space $(\mathbb{N} \cup \{0\})^{|V|-1}$ which has already achieved stationary distribution π , i.e., $Q_0(v) = 0$ for all $v \in V \setminus \{u_s\}$ whereas $\bar{Q}_0(v)$ is non-zero in general.

Now, consider the coupling (Q_t, \bar{Q}_t) on $(\mathbb{N} \cup \{0\})^{|V|-1} \times (\mathbb{N} \cup \{0\})^{|V|-1}$ defined over random sequences $\{0, 1\} \times \{\prod_{v \in V \setminus \{u_s\}} \Gamma(v)\}$ where $\Gamma(v)$ is the set of one-step destinations from node v , such that the chain Q_t starts with empty queues i.e., $Q_t(u) = 0, \forall u \in V \setminus \{u_s\}$ and the queues in chain \bar{Q}_t are populated according to stationary distribution π . Such Markov chains are said to be stochastically ordered chains in the queueing theory and have a property that the Markov chain which dominates the other chain will always maintain dominance over it.

Now, under this coupling we allow the two chains to run in a way that any data generation or data transmission decision made by any queue in one chain is followed by the corresponding queue in the other chain as well. However, suppose the packets already in \bar{Q}_t are distinguished from the newly generated packets and the latter ones get a preference in the transmission. Given that η_u^t is the queue occupancy probability of node u in the chain Q_t and η_u is its steady-state queue occupancy probability in Markov chain \bar{Q}_t . Then, we know $\eta_u^t \leq \eta_u$ from Claim 1. To ensure

both chains get coupled all the old packets in \bar{Q}_t need to be sunk. However by our preference in transmission, the probability that such packets move out of queue in one time step is equal to the probability that corresponding queue in Q_t is empty i.e., $1 - \eta_u^t$. Also, we have $1 - \eta_u^t \geq 1 - \eta_u \geq \min_u 1 - \eta_u \geq 1 - \eta_{max}$. So, probability of packet moving out of queue in \bar{Q}_t in t_{exp} time steps is at least $t_{exp}(1 - \eta_{max})$. Now, since we know the sink collects data packets at rate β , so in t_{exp} time steps we have $\beta t_{exp}(1 - \eta_{max})$.

Now, if we consider the expected number of packets already residing in the queues of Markov chain \bar{Q}_t in the beginning (as per the stationary distribution), we know it is equal to the product of expected latency of a data packet to reach the sink and the rate of collection by the sink. To bound the expected latency we have the following claim.

Claim 2. *Given a data collection process on a graph $G = (V, E, w)$ such that $\beta < \beta^*$, the expected time spent in G by a data packet before it gets sunk is $\frac{t_{hit}}{1 - \eta_{max}}$ where t_{hit} is the worst-case hitting time of random walk on graph G and η_{max} is the maximum queue occupancy probability of all nodes in $V \setminus \{u_s\}$ at stationarity.*

The proof of this claim follows by first analysing the time taken by a data packet to hit the sink without any queueing delays which in the worst-case is t_{hit} and then combining it with the delay which is at most η_{max} .

Now, to use this claim we know that the sink collects data packets at rate β , so in t_{exp} time steps we have $\beta t_{exp}(1 - \eta_{max}) = \frac{\beta t_{hit}}{1 - \eta_{max}}$ which gives us $t_{exp} = \frac{t_{hit}}{(1 - \eta_{max})^2}$ where t_{exp} is the time by which all old packets in \bar{Q}_t have sunk i.e., the expected time by which the two chains couple. To bound this time, we cannot operate very close to the critical data rate β^* , so we operate at $3\beta^*/4$, a data rate which is constant factor away from the critical data rate. To use this value, let us first prove an important property of η with respect to data rate β . Given a $\beta' < \beta^*$, we have from Eq. (7) $\eta'^T(I - \mathcal{P}) = \beta' \mathbf{J}^T$. Multiplying both sides of this equation by β^*/β' we have $\frac{\beta^*}{\beta'} \eta'^T(I - \mathcal{P}) = \beta^* \mathbf{J}^T$. This gives us $\eta^* = \frac{\beta^*}{\beta'} \eta'$ i.e., η is linear in β . From this property and the fact that at β^* , the maximum queue occupancy probability $\eta_{max} = 1$, operating at a data rate of $3\beta^*/4$ gives us $t_{exp} = 16t_{hit}$.

The distance of a Markov chain from the stationarity is known to be related to the expected coupling time. We use the formulation of this property as presented by Levin, Peres and Wilmer [15]:

Lemma 7 (Corollary 5.4, Levin et al. [15]). *Let $\{(X_t, Y_t)\}$ be a coupling with initial states $x, y \in \mathcal{X}$ such that $X_0 = x$ and $Y_0 = y$ and coupling time defined as $\tau_{couple} := \min\{t : X_s = Y_s \text{ for all } s \geq t\}$, then,*

$$d(t) = \max_{x \in \mathcal{X}} \|\mathcal{P}^t(x, \cdot) - \pi\|_{TV} \leq \max_{x, y \in \mathcal{X}} P_{x, y}\{\tau_{couple} > t\}$$

where π is the stationary distribution.

So, we know $\max_{x, y} E_{x, y}(\tau_{couple}) = 16t_{hit}$. So, from Lemma 7 and Markov's inequality we have

$$d(t) = \max_{x \in \mathcal{X}} \|\mathcal{P}^t(x, \cdot) - \pi\|_{TV} \leq \frac{\max_{x, y} E_{x, y}(\tau_{couple})}{t} = \frac{16t_{hit}}{t} \quad (14)$$

Now, from the definition of mixing time we know $t_{mix}(\epsilon) = \min\{t : d(t) \leq \epsilon\}$ and $t_{mix}(\epsilon) = t_{mix} \log \epsilon^{-1}$ where $t_{mix} = t_{mix}(1/4)$. Now, let us bound the time by which our Markov chain Q_t is ϵ away from its stationary distribution i.e., $\|\mathcal{P}t(0, \cdot) - \pi\|_{TV} \leq \epsilon$. So from Eq. 14 and definition of mixing time we have

$$t_{mix}(\epsilon) = 64t_{hit} \log \epsilon^{-1}. \quad (15)$$

□

Now, let us use this lemma to prove the correctness of our subroutine DRW-Compute for $\beta < \beta^*$.

Proof of Lemma 3. Given a data rate $\beta < \beta^*$, our subroutine computes the steady-state queue occupancy probability of nodes by first running the Markov chain Q_t defined on the queue size of nodes $u \in V \setminus \{u_s\}$ close to the stationary distribution. After the chain is close to its stationarity, then it starts sampling the values of its queue occupancy.

Firstly, let us suppose Markov chain Q_t is ϵ_1 distance from the stationary distribution π . Using Lemma 6 we can bound this time, let this be t_1 . Now, given that all nodes are initially empty for $\|\mathcal{P}^t(0, \cdot) - \pi\|_{TV} \leq \epsilon_1$ we have $t_1 = 64t_{\text{hit}} \log \epsilon_1^{-1}$. Now, given a constant $\epsilon_2 > 0$ we have $\eta_u(1 - \epsilon_1)\epsilon_2 \geq 0$, so using Hoeffding's inequality we have

$$\begin{aligned} P[|\bar{X}(u) - \mathbb{E}[\bar{X}(u)]| \geq \eta_u(1 - \epsilon_1)\epsilon_2] &\leq 2 \exp\left(\frac{-2t_u'^2(\eta_u(1 - \epsilon_1)\epsilon_2)^2}{t_u'}\right), \\ &\leq 2 \exp\left(-2(t_u'\eta_u^2(1 - \epsilon_1)^2\epsilon_2^2)\right). \end{aligned} \quad (16)$$

So, after sampling for time t_u' with high probability we get $(1 - \epsilon_1 - \epsilon_2)$ estimate of steady-state queue occupancy probability η_u when

$$t_u' = \frac{\log n}{\eta_u^2(1 - \epsilon_1)^2\epsilon_2^2}.$$

or t_u' is greater than the given term. To bound this term we know $\epsilon_1 \leq 1/2$. So, we have $t_u' = \frac{4 \log n}{\eta_u^2 \epsilon_2^2}$. Moreover, as we are sampling component wise (separately for each node $u \in V \setminus \{u_s\}$), so our overall sampling time will be

$$t_2 = \max_u t_u' \geq \max_u \frac{4 \log n}{\eta_u^2 \epsilon_2^2}. \quad (17)$$

So, we get the overall time as $t_1 + t_2 \geq 64t_{\text{hit}} \log \epsilon_1^{-1} + \max_u \left(\frac{4 \log n}{\eta_u^2 \epsilon_2^2}\right)$. Now, to bound this time we consider all nodes $u \in V \setminus \{u_s\}$ such that their node potentials satisfy $\kappa < \eta_u$ where $0 < \kappa < 1$. So, using this fact our subroutine computes the estimates $\hat{\eta}_u, \forall u \in V \setminus \{u_s\}$ such that

$$|\hat{\eta}_u - \eta_u| \leq (\epsilon_1 + \epsilon_2)\eta_u, \forall \kappa < \eta_u \quad (18)$$

in time $\left(64t_{\text{hit}} \log \epsilon_1^{-1} + \frac{4 \log n}{\kappa^2 \epsilon_2^2}\right)$. □

3.3.2 Analyzing DRW-Compute for $\beta \geq \beta^*$

For the case of unstable data rates $\beta \geq \beta^*$, we know the Markov chain is non-ergodic. However, our subroutine can still return the values which can serve as an indicator for identifying such data rates. In particular, we have the following result.

Proof of Lemma 4. To understand the working of our subroutine for data rates $\beta \geq \beta^*$, let us first consider the following property of queue occupancy probability.

Claim 3. *Given a data collection process with source nodes having an independent Bernoulli data generation with rate β . Let Q_t^β represent the queues at time t for all nodes $u \in V \setminus \{u_s\}$. Then, for all such nodes $P[Q_t^\beta(u) > 0]$ is an increasing function of β .*

This claim can be proved by first showing the stochastic orderedness property of Markov chain Q_t^β (see Claim 1) and then proving the monotonicity in β using a similar coupling.

Now, to use this claim let us consider a data rate close to the critical rate β^* i.e., $\beta' = \beta^* - \epsilon$. Since $\beta' < \beta^*$ from Claim 3 we have $P[Q_t^{\beta'}(u) > 0] < P[Q_t^{\beta^*}(u) > 0]$ i.e., $\hat{\eta}_u^{\beta'} < \hat{\eta}_u^{\beta^*}$. As, β' is stable we know from Lemma 3, our subroutine will return estimates $\hat{\eta}_u^{\beta'}$ in time $64t_{\text{hit}} \log \epsilon_1^{-1} + \frac{4 \log n}{\kappa^2 \epsilon_2^2}$ where t_{hit} is the worst-case hitting time of random walk on the underlying graph. So, after the given time we can compute $\hat{\eta}_{\max}^{\beta'} = \max_{u \in V \setminus \{u_s\}} \hat{\eta}_u^{\beta'} = 1 - (\epsilon_1 + \epsilon_2 + \epsilon)$. Now, since $\hat{\eta}_u^{\beta'} < \hat{\eta}_u^{\beta^*}$ and in fact η is linear in β , in same time for $\beta \geq \beta^*$ solver will return $\hat{\eta}_u^{\beta^*}$ such that $\hat{\eta}_{\max}^{\beta^*} = \max_{u \in V \setminus \{u_s\}} \hat{\eta}_u^{\beta^*} \geq 1 - (\epsilon_1 + \epsilon_2)$. \square

Remark: As discussed before, instead of operating at data rates very close to the critical data rate, we opt for rate $3\beta^*/4$. So for all data rates $\beta \geq 3\beta^*/4$ from Lemma 4 we have $\hat{\eta}_{\max} \geq 3/4(1 - (\epsilon_1 + \epsilon_2))$. This acts as a test condition for checking the stability of data rates which is used by our solver DRW-LSolve to compute a stable data rate for computation operation as discussed in Section 3.1.

3.3.3 Effective resistance computation

DRW-LSolve can be directly used to calculate the effective resistance $R_{u,v}$ between a pair of nodes $u, v \in V$. We discuss that result next.

Proof of Corollary 1. We know from the steady-state equation Eq. (9), the effective resistance between two nodes u and v is $R_{u,v} = \frac{x_u - x_v}{\beta}$. To compute this resistance $R_{u,v}$ let us assume u is the source node (denoted by s) and v is the sink (denoted by u_s). Since, we always assume queue occupancy probability of sink to be zero so to compute the effective resistance we only need to compute η_s accurately. Using Theorem 1 we know DRW-LSolve can compute estimates $\hat{\eta}_u$ such that $|\hat{\eta}_u - \eta_u| \leq (\epsilon_1 + \epsilon_2)\eta_u, \forall \kappa < \eta_u$ in time $\left(64t_{\text{hit}} \log \epsilon_1^{-1} + \frac{4 \log n}{\kappa^2 \epsilon_2^2} + 2\text{diam}(G)\right) \log \frac{16d_{\max}}{3\lambda_2^2}$. In this case, since u is the only source and v is the sink, the expected latency is $t_{\text{hit}}(u, v)$ instead of t_{hit} . Also, we can ignore the $\text{diam}(G)$ factor in running time which comes from the exchange of messages between the controller node and other nodes, by making the source node u as the controller. In such a scenario, for each binary search iteration source node will need to run DRW-Compute locally as well.

Now, to accurately compute η_s we need to set value of κ appropriately. For this consider the harmonic property of potential x . By this property, we know that the potential at source node x_s is the maximum. So, we have $x_s > x_u, \forall u$. As, $x_u = \eta_u/d_u$, we have $\frac{\eta_s}{d_s} > \frac{\eta_u}{d_u}$ for all u . This gives

$$\eta_s > \frac{d_s}{d_u} \eta_u > \frac{d_{\min}}{d_{\max}} \eta_u \quad (19)$$

Moreover, since the lowest value of β that DRW-LSolve can reach through binary search is $3\beta^*(1 - (\epsilon_1 + \epsilon_2))/8$, so at that rate there exists a node u^* whose queue occupancy probability is maximum i.e., $3(1 - (\epsilon_1 + \epsilon_2))/8$. Now, if we consider $\epsilon_1 + \epsilon_2 < 1/2$, we know $\eta_{u^*} > 3/16$. Using this value in Eq. (19) we have $\eta_s > \frac{3d_{\min}}{16d_{\max}}$. So, for correct estimation of η_s we can set the value of κ as $\frac{3d_{\min}}{16d_{\max}}$. So, we get the overall time for effective resistance computation $R_{u,v}$ as $\left(64t_{\text{hit}}(u, v) \log \epsilon_1^{-1} + \left(\frac{16d_{\max}}{3d_{\min}}\right)^2 \frac{4 \log n}{\epsilon_2^2}\right) \log \frac{16d_{\max}}{3\lambda_2^2}$. \square

3.4 Discussion: The CONGEST Model

For modeling communication in a data collection process we use the standard CONGEST model [20]. In this model, the underlying network is modeled as a graph $G = (V, E, w)$ and the communication is done in discrete synchronous rounds. In each such round, every node can send $O(\log n)$ size message to all its neighbor. However, in our case we do not use the full power of the CONGEST model since a node picks at most one neighbor to send a packet to in a given round.

To see that the message size is $O(\log n)$ we note that whenever the subroutine DRW-Compute is run it starts from scratch and runs for a number of time steps which is polynomial in n and in the inverse of the error parameters (see the statements of Lemmas 3 and 4). Hence when a source, say node u , generates a packet it can give it id $\langle u, \text{sequence number} \rangle$ where the sequence numbers begin from 0 and increment for every new packet. Since the number of time steps is polynomial in n , as long as the error parameters are inverse polynomials in n the id of each packet is $O(\log n)$ bits in length.

4 Distributed Generation of Random Spanning Trees

In this section, we present an application of our distributed Laplacian solver to generate approximately uniform random spanning trees. We first begin by formally defining our problem of random spanning tree generation and then we present a distributed algorithm to solve it which uses our solver DRW-LSolve as a subroutine. Our proposed algorithm is basically a distributed version of Kelner and Mądry's [13] algorithm. So, we will first give an overview of their approach and also discuss how we adopt it in a distributed setting. Then, we will discuss our algorithm in detail indicating where our distributed solver will be used and then finally we would review the overall complexity of our algorithm.

4.1 Random Spanning Tree Generation

Given an undirected graph $G = (V, E)$ with $|V| = n$ vertices and $|E| = m$ edges each having unit weight, the random spanning tree generation problem requires us to find an algorithm which outputs a spanning tree T of G with probability $1/T(G)$, where $T(G)$ is the set of all spanning trees of G . However, in this paper we look at a relaxed version of this problem, known as ϵ -random spanning tree generation wherein we find an algorithm which generates each spanning tree T with a probability $P[T]$ such that $\frac{(1-\epsilon)}{T(G)} \leq P[T] \leq \frac{(1+\epsilon)}{T(G)}$ i.e., probability of generation is ϵ -away from the uniform distribution. Our proposed algorithm generates these ϵ -random spanning trees in a distributed setting. In particular, our algorithm is a distributed version of Kelner and Mądry's [13] algorithm based on the famous result of generating uniformly random spanning trees using random walks by Broder [8] and Aldous [3]. We give an overview of our approach and compare it with that of Kelner and Mądry next.

4.2 Comparative Overview of Kelner and Mądry's Algorithm and our Approach

Kelner and Mądry [13] use random walk-based algorithms for random spanning tree generation. Their approach is based on the result of Broder [8] and Aldous [3] who independently showed that if we simulate a random walk on a graph starting from an arbitrary vertex and continue till all vertices are visited, then the set of edges through which each vertex was visited first time by the given walk forms a uniformly random spanning tree of the given graph. However, since this would take time equivalent to the cover time of the graph which can be $O(mn)$ in expectation. Kelner and

Mađry proposed an algorithm to simulate this random walk more efficiently. They observed that the random walk spent a lot of time revisiting the vertices of the graph, so simulating those portions was wasteful. They used the standard ball-growing technique of [14] to decompose the graph into low diameter partitions which could be quickly covered by the random walk. Then, *for each partition using Laplacian solvers [26] they precompute the approximate probability of random walk entering that partition from a particular vertex and exiting from a particular vertex as these correspond to the potential developed at the vertex if the given partition is assumed to be an electrical network and the exit vertex is assumed to be attached to a voltage source with a dummy vertex added to the partition assumed to be the sink.* Further, this approximate exit distribution helps them to shortcut the random walk by removing its trajectories after all vertices of the partition have been visited. Finally, they use these precomputed values to simulate random walk on the graph and generate ϵ -random spanning trees.

Our proposed algorithm Distributed RST Generation is also based on a similar approach of generating random spanning trees using random walks on the graph. However, we give a completely distributed algorithm for the problem. We first use a distributed version of Miller et al.'s [17] algorithm for decomposition of graph into low diameter partitions (S_1, \dots, S_k) and set C of edges not entirely contained in one of S_i . Miller et al.'s graph decomposition is a parallel version of ball-growing technique with random delays. In this, each node selects a random start time according to some distribution, and if a node is not already part of a partition at that time, it begins its own breadth first search (BFS) to form its cluster. Any node visited by the search which is not part of any partition joins the partition of the node which reached it first and accordingly adds its neighbors to the corresponding BFS queue. Delayed and random start times ensure that the partitions have the desired properties required from the decomposition. We make this algorithm distributed by exchanging messages among the nodes so that all the nodes know the random start times of all other nodes and also use a distributed version of BFS [6][11]. Moreover, each node which starts the partition is made the leader of the corresponding partition and is responsible for exchange of synchronization messages among various clusters. *Once these partitions are made then instead of using Laplacian solver we use our distributed solver to compute the approximate exit distribution for each partition. So, we run DRW-LSolve in parallel in each partition S_i to compute the $(1 + \epsilon)$ -approximation of exit distribution $P_v(e)$ i.e., probability of entering partition through vertex v and exiting from edge e where $v \in V(S_i)$ (vertex set of S_i), $e \in C(S_i)$ (set of edges in C with one end-point in S_i).* In addition, we run a random walk on each partition in parallel to compute the spanning tree \hat{T}_{S_i} within each S_i using Broder and Aldous result. The completion of computation step by all S_i 's is indicated by exchange of synchronization messages among the leader nodes. After this step, we reduce graph G to G' such that each S_i is assumed to be a super node and we combine all edges (u, w) where $u \in V(S_i)$ and $w \in V(S_j)$ into a super edge connecting two partitions. We then run a random walk on G' with transition probability $\mathcal{P}[S_i, S_j] = \frac{\sum_{v \in S_i} \sum_{u \in S_i} \sum_{w \in S_j} P_v(u, w)}{\sum_{v \in S_i} \sum_{u \in S_i} \sum_{w \in S_j} P_v(u, w)}$ where $P_v(u, w)$ are the computed exit distributions representing the probability of a random walk entering a partition S_i through vertex $v \in V(S_i)$ and exiting through edge $e = (u, w) \in C(S_i)$. After that, again using the Broder and Aldous result we obtain spanning tree $\hat{T}_{G'}$ on the reduced graph G' . However, note that within each super node random walk takes a predetermined path from the entry vertex v to exit edge e and information about it is exchanged among the nodes in the given partition. Finally, by combining the spanning tree within S_i 's i.e., \hat{T}_{S_i} to that of the reduced graph $\hat{T}_{G'}$ we obtain ϵ -random spanning tree of the given graph \hat{T}_G . Refer to Figure 2 for the explicit changes made to the Kelner and Mađry's algorithm to adapt it to a distributed setting.

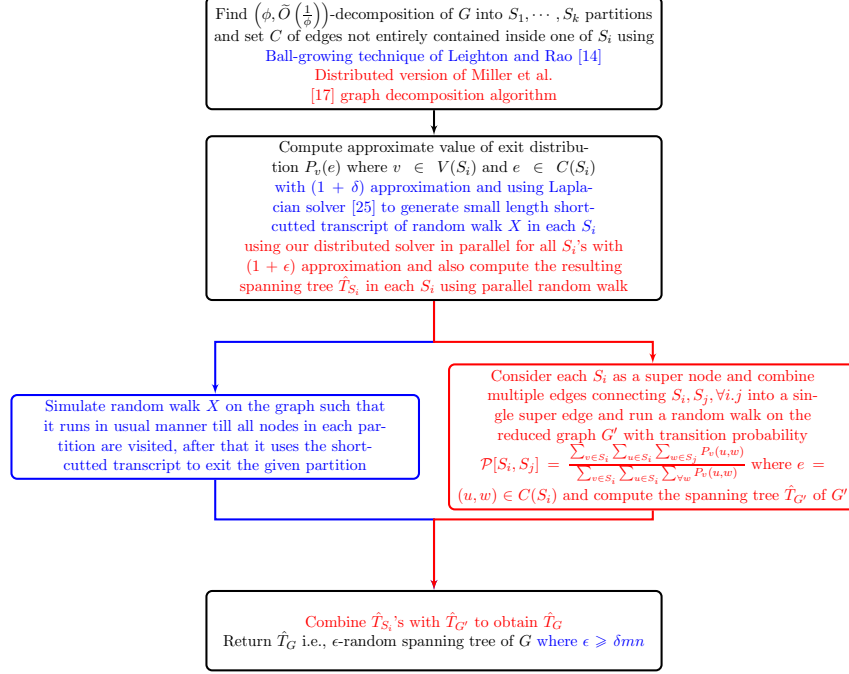


Figure 2: Comparison of Kelner and Mądry's [13] Algorithm and our Distributed RST Generation Algorithm for random spanning tree generation (black color denotes common components).

4.3 Distributed RST Generation Algorithm

Having defined our distributed approach for random spanning tree generation and how it differs from Kelner and Mądry's approach, let us now discuss it in detail. As discussed before, our approach is based on the famous result of Broder and Aldous of generating random spanning trees using random walks. In particular, our proposed algorithm Distributed RST Generation has three main steps: first of which requires a low-diameter decomposition of given graph into partitions and the last two use Aldous, Broder result with random walks as a basic primitive and our distributed solver as a subroutine to precompute exit distributions for each partition. Let us discuss each of these steps.

Graph decomposition The first step of our proposed algorithm is to decompose our given graph into low-diameter partitions which can be easily processed. Let us first formally define the decomposition we require for our algorithm.

Definition 2 ((ϕ, γ) -decomposition). *Given a graph $G = (V, E)$ a (ϕ, γ) -decomposition splits it into partitions (S_1, \dots, S_k) and set C of edges not entirely contained in one of the partitions S_i such that*

- The diameter of each S_i i.e., $\gamma(S_i)$ is at most γ , and
- $|C| \leq \phi |E(G)|$.

Kelner and Mądry use ball-growing technique [14] to obtain (ϕ, γ) -decomposition of graph. However, we will use Miller et al.'s [17] algorithm in a distributed setting to obtain given decomposition of G . In Miller et al.'s algorithm, a random shift δ_u is picked for all nodes from independent exponential distribution with parameter ϕ . After that, each node is assigned to a partition such

Algorithm Distributed RST Generation

Require: Unit-capacity graph $G = (V, E)$, error parameter ϵ , and $\phi \in (0, 1)$

- 1: Run Distributed Decomposition (ϕ) to obtain $(\phi, \tilde{O}(\frac{1}{\phi}))$ -decomposition of G , and set L
/* Such that G is decomposed into S_1, \dots, S_k partitions and set C of edges not entirely contained in one of S_i , L is set of leader nodes for all S_i */
 - 2: Set vector \mathbf{b} such that $\mathbf{b}_1 = 1$, $\mathbf{b}_{n_i} = -1$, and $\mathbf{b}_i = 0$ for $i \neq \{1, n_i\}$ /*where $n_i = |V(S_i)|$ */
 - 3: Each S_i runs DRW-LSolve $(\frac{1}{\sqrt{n_i}}, \mathbf{1}, \mathbf{b}, \epsilon, 1/2)$ in parallel to compute $(1 + \epsilon)$ -approximation of exit distribution $P_v(e)$ where $v \in V(S_i)$, $e \in C(S_i)$ /* where $\mathbf{1}$ denotes unit weight $\forall e \in E(S_i)$ */
 - 4: In parallel, compute the spanning tree \hat{T}_{S_i} within each S_i using a random walk
 - 5: Leader nodes $i \in L$ of each partition S_i exchange messages to indicate end of their computation step
 - 6: **if** Messages from all $k - 1$ partitions received by all $i \in L$ /* Messages from everyone other than themselves */ **then**
 - 7: Reduce graph G to G' by considering each S_i as a super node and combine multiple edges connecting two partitions into a single super edge
 - 8: Run a random walk on the reduced graph G' with transition probability $\mathcal{P}[S_i, S_j] = \frac{\sum_{v \in S_i} \sum_{u \in S_i} \sum_{w \in S_j} P_v(u, w)}{\sum_{v \in S_i} \sum_{u \in S_i} \sum_{w \in S_j} P_v(u, w)}$ /* where $P_v(u, w)$ is the exit distribution for $e = (u, w) \in C(S_i)$ */
 - 9: Set of first visited edges by the random walk on G' forms its ϵ -random spanning tree $\hat{T}_{G'}$
 - 10: **end if**
 - 11: Combine \hat{T}_{S_i} of all S_i with $\hat{T}_{G'}$ to obtain \hat{T}_G /* where \hat{T}_G is ϵ -random spanning tree of G */
 - 12: **return** \hat{T}_G
-

Algorithm Distributed Decomposition (ϕ) (Distributed version of Miller et al. [17])

Require: Unit-capacity graph $G = (V, E)$, parameter $0 < \phi < 1$

- 1: In parallel, each vertex u picks δ_u independently from an exponential distribution with mean $1/\phi$
 - 2: In parallel, each node computes $\delta_{max} = \max\{\delta_u : u \in V\}$ by exchanging δ_u values
 - 3: Perform Distributed BFS [11], with vertex u starting when the vertex at the head of the queue has distance more than $\delta_{max} - \delta_u$
 - 4: Add vertex u to set L /* u is starting vertex of a partition, L is set of leader nodes for partitions */
 - 5: In parallel, each vertex u assigns itself to the point of origin of the shortest path that reached it in the BFS
 - 6: **return** $(\phi, \tilde{O}(\frac{1}{\phi}))$ -decomposition of graph and set L
-

that the distance $\delta_{max} - \delta_u$ is minimized where $\delta_{max} = \max_u \delta_u$. The clusters which will represent our partitions are created using breadth first search (BFS) i.e., if a node u is not already part of a partition by its chosen start time δ_u then, it starts its own partition by performing a BFS, otherwise the node joins the partition that reached it first. We make this algorithm distributed by exchange of messages among the nodes and using distributed version of BFS [6][11]. Moreover, each node which starts the partition is designated to be the leader of that partition and is responsible for exchange of messages on behalf of its partition. The randomized start times chosen by the nodes ensure that the required properties of the (ϕ, γ) -decomposition are satisfied.

First to bound the diameter of partitions, Miller et al. [17] bound the distance between a node and the leader of the partition to which it is assigned. Since the chosen shift value δ_u of the leader of partition S_u bounds the distance to any node in S_u , so $\delta_{max} = \max_u \delta_u$ is an upper bound on the diameter of each partition. The following lemma gives the bound on the maximum shift value and hence, the diameter of each partition.

Lemma 8 (Lemma 4.2, Miller et al. [17]). *Given that each node $u \in V$ chooses a random shift value δ_u from an exponential distribution with parameter ϕ , the expected value of the maximum shift value δ_{max} is given by H_n/ϕ where H_n is the n th harmonic number. Furthermore, with high probability, $\delta_u \leq O\left(\frac{\log n}{\phi}\right)$ for all u .*

For the other property which requires that there are fewer edges between the partitions, Miller et al. show this by bounding the probability that the endpoints of an edge are assigned to different partitions. In particular, Miller et al. prove the following lemma.

Lemma 9 (Corollary 4.5, Miller et al. [17]). *Given a (ϕ, γ) -decomposition of graph $G = (V, E)$ with $|V| = n$ nodes and $|E| = m$ edges into partitions (S_1, \dots, S_k) , the probability of an edge $e = (uv) \in E$ having $u \in V(S_i)$ and $v \in V(S_j)$ such that $S_i \neq S_j$ is bounded by $O(\phi)$, and the expected number of edges between the partitions is $O(\phi m)$.*

Both these lemmas will hold for our distributed version as well because we only differ from Miller et al. in the way δ_{max} is computed and BFS is performed by the nodes.

Computation within partitions Now, given that we have low diameter partitions of the graph, we will use them along with the following famous result by Broder, Aldous for ϵ -random spanning tree generation.

Lemma 10 (Broder [8], Aldous [3]). *Given an undirected graph $G = (V, E)$ suppose you start a random walk from an arbitrary vertex $u \in V$ and let T be the set of edges used by the walk for the first visit to each vertex $v \in V \setminus \{u\}$, then T forms a uniformly random spanning tree of G .*

We know for our low diameter partitions random walk will take less time to visit all the vertices in it, however, after all the first visits it may still spend a lot of time visiting already covered regions. To avoid these unnecessary steps we need to somehow shortcut the random walk once all nodes in a region are visited. Kelner and Mądry [13] suggested to compute exit distributions from a given partition $P_v(e)$ where $v \in V(S_i)$ and $e \in C(S_i)$ i.e., the probability of random walk leaving partition S_i through edge e after it had entered it through vertex v . Given a (ϕ, γ) -decomposition of graph they compute $(1 + \epsilon)$ -approximation of all $P_v(e)$ using Laplacian solvers [21]. In particular, they use the following construction for $P_v(e)$ computation: Given a partition S_i and edge $e = (u, u') \in C(S_i)$ with $u \in V(S_i)$ they construct S'_i by adding vertex u' and some dummy vertex u^* to S_i . Then, for each boundary edge $(w, w') \in C(S_i) \setminus \{e\}$ with $w \in V(S_i)$, they add an edge (w, u^*) and finally add edge $e = (u, u')$ to it. After the given construction, they treat S'_i as an electric

circuit and impose voltage of 1 at u' and 0 at u^* and then using Laplacian solvers compute the approximate electrical flow in it wherein the voltage achieved at any node $v \in V(S_i)$ is equal to $P_v(e)$ (see Lemma 9 [13] for details). We will also use a similar construction, however, to compute these flows and the resulting potentials we will use our distributed solver and we will compute these values in parallel in all partitions. Moreover, at the same time we will run a random walk in each partition in parallel so that from Lemma 10 we obtain uniform spanning tree for each S_i . As we will discuss in detail in Section 4.4, by setting $\phi = 1/\sqrt{n}$ in each partition S_i , by the time random walk-based solver computes $P_v(e)$ for all $e \in C(S_i)$ the parallel random walk would have covered the entire partition, hence, giving us the desired uniformly random spanning tree for S_i along with the exit distributions.

ϵ -random spanning tree generation Once all partitions have computed their spanning trees and exit distributions, their respective leaders exchange synchronization messages to indicate the completion of computation step. After messages from all k partitions are received by the leader nodes, we have obtained uniform spanning tree \hat{T}_{S_i} for all S_i 's as well as their exit distributions. So, to obtain random spanning tree for G we need to find edges between the different \hat{T}_{S_i} 's. For this we reduce graph G to G' such that each partition S_i represents a super node and multiple edges connecting two partitions are combined into a single super edge i.e, we combine all edges (u, w) where $u \in V(S_i)$ and $w \in V(S_j)$. We then use the computed exit distributions of each partition to run a random walk on G' with transition probability $\mathcal{P}[S_i, S_j] = \frac{\sum_{v \in S_i} \sum_{u \in S_i, w \in S_j} P_v(u, w)}{\sum_{v \in S_i} \sum_{u \in S_i, \forall w} P_v(u, w)}$ where $P_v(u, w)$ is the probability of a random walk entering a partition S_i through vertex $v \in V(S_i)$ and exiting through edge $(u, w) \in C(S_i)$. However, note that within each such super node S_i , the random walk will take a predetermined path based on its spanning tree inside S_i and thus would need to exchange messages between nodes and the leader of partition. So, each such step would take $O(\text{diam}(S_i))$ time where $\text{diam}(S_i)$ is the diameter of partition S_i . After the cover time of this random walk, since all vertices of G' will be visited we will obtain random spanning tree $\hat{T}_{G'}$ for G' . Finally, combining this random spanning tree with that of all S_i 's, we obtain ϵ -random spanning tree of G .

4.4 Overall Complexity

Now, let us review the overall complexity of our algorithm. Our proposed algorithm has three main parts: decomposing graph into low diameter partitions, then using random walk on those partitions to obtain spanning trees within them and finally finding the edges between the partitions to compute the overall spanning tree. For the first part we compute $(\phi, \tilde{O}(1/\phi))$ -decomposition of the graph into (S_1, \dots, S_k) partitions using distributed version of Miller et al.'s algorithm which takes about $O(m + \text{diam}(G))$ time where factor $\text{diam}(G)$ (diameter of graph) comes from the distributed BFS and message exchanges and factor m comes from verifying the decomposition. Then, for each partition S_i we compute the exit distribution which corresponds to the node potentials using our proposed distributed solver which computes the estimates of node potentials $\hat{x}_u, \forall u \in V(S_i)$ such that $|\hat{x}_u - x_u| \leq (\epsilon_1 + \epsilon_2)x_u, \forall \kappa < x_u d_u$ where $0 < \kappa < 1$ and $x_u = \eta_u/d_u$ in time

$$\left(64t_{\text{hit}}^{S_i} \log \epsilon_1^{-1} + \frac{4 \log n_i}{\kappa^2 \epsilon_2^2} + 2\text{diam}(S_i) \right) \log \frac{16d_{\max}}{3\lambda_2^L}$$

where $t_{\text{hit}}^{S_i}$ is the worst-case hitting time of random walk on partition S_i with $|V(S_i)| = n_i$ as the size of the partition. As $\text{diam}(S_i)$ cannot be greater than the worst-case hitting time over this partition, so this time reduces to $\tilde{O} \left(t_{\text{hit}}^{S_i} \log \epsilon_1^{-1} + \frac{\log n_i}{\kappa^2 \epsilon_2^2} \right)$.

Now, let us consider $\phi = 1/\sqrt{n}$, so from the first step we obtain a $(1/\sqrt{n}, \tilde{O}(\sqrt{n}))$ -decomposition of the graph into partitions (S_1, \dots, S_k) . Moreover, within each partition S_i to compute the node potentials accurately we can set the values of $\kappa = 1/\sqrt{n_i}$ and $\epsilon_2 < 1/2$. Also, since the worst-case hitting time for any graph is $\Omega(n)$ [2] we get the running time of solver for each edge $e \in C(S_i)$ in partition S_i as $\tilde{O}(t_{\text{hit}}^{S_i} \log \epsilon^{-1})$ where ϵ is the error. This is repeated for all edges in set $C(S_i)$ (subset of C incident to S_i), so the total time taken for computation of exit distribution for a given partition S_i is $\tilde{O}(|C(S_i)| (t_{\text{hit}}^{S_i} \log \epsilon^{-1})) \leq \tilde{O}(\phi m (t_{\text{hit}}^{S_i} \log \epsilon^{-1}))$, as from (ϕ, γ) -decomposition $|C| \leq \phi m$. Moreover, the random walk that we run in parallel in this partition to find its spanning tree will take at most cover time $t_{\text{cov}}(S_i)$ to visit all vertices. So, for each partition S_i , time taken to compute the exit distributions and the spanning tree is $\max \left\{ \tilde{O}(\phi m (t_{\text{hit}}^{S_i} \log \epsilon^{-1})), t_{\text{cov}}(S_i) \right\}$.

From Aleliunas et al. [4] we know that the cover time of an unweighted graph G with diameter $\text{diam}(G)$ is at most $O(|E(G)| \text{diam}(G))$, so for S_i we have $t_{\text{cov}}(S_i) \leq m_i \sqrt{n}$ as the diameter of each partition is at most \sqrt{n} . As, the worst-case hitting time of graph is greater than that of its partition i.e., $t_{\text{hit}}^{S_i} \leq t_{\text{hit}}$, we have $\tilde{O}(\phi m (t_{\text{hit}}^{S_i} \log \epsilon^{-1})) \leq \tilde{O}(\phi m (t_{\text{hit}} \log \epsilon^{-1}))$. So, by the time our random walk-based solvers compute the exit distributions, the random walk running in parallel has covered all vertices to give us the random spanning tree of each S_i . Now, since we do this computation step in parallel for all partitions, we have the overall time as $\max_i \left\{ \max \left\{ \tilde{O}(\phi m (t_{\text{hit}} \log \epsilon^{-1})), t_{\text{cov}}(S_i) \right\} \right\} \leq \tilde{O}(\phi m (t_{\text{hit}} \log \epsilon^{-1}))$. Moreover, after each partition completes the computation step its leader exchanges synchronization messages with other leader nodes which takes about $O(k \text{diam}(G))$ time where $\text{diam}(G)$ is the diameter of the graph and k are the total number of partitions. Once all partitions know that the computation step is over and they proceed to the last step. In the final step, we run the random walk on the reduced graph G' and it takes at most $t_{\text{cov}}(G') = O(k^3)$ to form the spanning tree where k is the number of partitions. However, since in each step of this walk within the super nodes we need to communicate entry and exit points of the partition which takes at most $\text{diam}(G)(S_i) \leq \sqrt{n}$ time, so overall time for this step is $O(k^3 \sqrt{n})$. Now, we know from the property of our (ϕ, γ) -decomposition that $|C| \leq \phi |E|$ and we have chosen $\phi = 1/\sqrt{n}$. Also, $|E| \leq n d_{\text{max}}/2$ where d_{max} is the maximum degree of graph G . So, to ensure that the graph is connected we have $k \leq \sqrt{n} d_{\text{max}}/2$. Thus, for bounded-degree graphs $k = O(\sqrt{n})$. So, our total time for Distributed RST Generation is composed of

- Decomposition of graph into low diameter partitions = $O(m + \text{diam}(G))$.
- Using random walks to compute spanning trees within those partitions = $\tilde{O}(\phi m (t_{\text{hit}} \log \epsilon^{-1}))$.
- Computing the overall spanning tree
 - Exchange of synchronization messages between leaders = $O(k \text{diam}(G))$.
 - Cover time of random on the reduced graph = $O(k^3 \sqrt{n})$.

As discussed above, since $\phi = 1/\sqrt{n}$, $k = O(\sqrt{n})$ for bounded-degree graphs, and $\text{diam}(G) \leq n$, we have the overall time as $O(m) + \tilde{O}\left(\frac{m}{\sqrt{n}} t_{\text{hit}} \log \epsilon^{-1}\right) + O(\text{diam}(G) \sqrt{n}) + O(n^2) = \tilde{O}\left(\frac{m}{\sqrt{n}} t_{\text{hit}} \log \epsilon^{-1}\right)$.

5 Related Work

After the breakthrough paper by Spielman and Teng [26] wherein the Laplacian equations are approximately solved in $\tilde{O}(m \log^c n \log 1/\epsilon)$ time where c is a constant and ϵ denotes the error, an extensive literature has been developed (see survey by [25], [27]) in which a number of quasi-linear time solvers have been proposed each improvising the value of exponent c in the running

time. However, most of these rely on multiple graph theoretic constructions and random sampling, making them difficult to analyze and implement. These are typically centralized algorithms.

In [22], Peng and Spielman gave the first parallel Laplacian solver, however because of the shared memory model the algorithm is not distributed. Similarly, a method that is amenable to being parallelized is that of Becchetti et. al. [7] who use a token diffusion process similar in spirit to our method to present a solver for the specific case of the electrical flow Laplacian. Their method involves simulating a large number of random walks on the network and cannot be adapted easily to the distributed setting since the number of tokens entering and leaving a vertex are potentially unbounded. Working towards a completely distributed algorithm, Zouzias and Freris [28], adapted the Kaczmarz iteration method for solving Laplacians [27, Chap. 14] to a gossip setting. Their approach is based on using the gossip model as a means of achieving consensus for solving Laplacians as a least-square estimation problem. As expected the convergence rate of their method depends linearly on the second smallest eigenvalue of the Laplacian matrix, λ_2^L , and the number of edges. Our method is completely different and is able to avoid the linear dependence on λ_2^L but with the disadvantage that our number of rounds depends on the hitting time of the natural random walk defined on the graph, which could be potentially $\Omega(m)$ but is $\Theta(n)$ for several important classes of graphs. Rebeschini and Tatikonda [23] analyze the performance of the message-passing min-sum algorithm to solve the electrical flow problem, which is a proper subset of the class of Laplacian systems we consider. Their findings are largely negative: they find that for most classes of graphs the min-sum algorithm is not able to converge to a solution, and identify one class for which the solution can be obtained in time proportional to the number of edges, as opposed to our algorithm that works for all connected graphs in time proportional to the hitting time of random walk which is $\Theta(n)$ for a large set of graphs.

The literature on electrical flow and its applications is too vast to survey here so we just mention that the Laplacian representation of the electrical flow problem has evolved it as a popular subroutine in solving various graph related problems like maximum flow computation [9], graph sparsification [25], random spanning tree generation [13]. We will show how our electrical flow computation method can be used to give a distributed version of the algorithm of Kelner and Mądry [13] which is the first distributed algorithm for the random spanning tree problem to the best of our knowledge, all prior work being centralized, e.g., the algorithms proposed in [19, 10, 24].

6 Conclusion and Future work

Although our main result presents a distributed algorithm, at a deeper level the key contribution of this paper is not the algorithm we present, but actually the connections our work makes with the queueing theory and ergodicity, and the theory of Markov chains and random walks. Positioning the Laplacian system in a network and solving it there connects the study of Laplacian systems to distributed systems and also raises the possibilities of real-world implementation in low-power networks like sensor networks where such problems are likely to occur naturally.

Moving this work ahead we plan to move beyond the one-sink constraint and investigate whether our methods extend to the entire class of Laplacian systems. We also feel that, although our current algorithm cannot be successfully adapted to better existing distributed algorithms for graph sparsification, it may be possible to adapt our methods to compete with or better the state of the art on this important problem.

References

- [1] N. Abreu, C. M. Justel, O. Rojo, and V. Trevisan. Ordering trees and graphs with few cycles by algebraic connectivity. *Linear Algebra and its Applications*, 458:429–453, 2014.
- [2] D. Aldous and J. Fill. Reversible markov chains and random walks on graphs, 2002.
- [3] D. J. Aldous. The random walk construction of uniform spanning trees and uniform labelled trees. *SIAM J. Discret. Math.*, 3(4):450–465, 1990.
- [4] R. Aleliunas, R. M. Karp, R. J. Lipton, L. Lovász, and C. Rackoff. Random walks, universal traversal sequences, and the complexity of maze problems. In *Proc. of the 20th Annual Symp. on Foundations of Computer Science*, SFCS '79, pages 218–223. IEEE Computer Society, 1979.
- [5] N. Alon, C. Avin, M. Koucký, G. Kozma, Z. Lotker, and M. R. Tuttle. Many random walks are faster than one. *Comb. Probab. Comput.*, 20(04):481–502, 2011.
- [6] B. Awerbuch and R. G. Gallager. Distributed BFS algorithms. In *Proc. of the 26th Annual Symp. on Foundations of Computer Science*, SFCS '85, pages 250–256. IEEE, 1985.
- [7] L. Becchetti, V. Bonifaci, and E. Natale. Pooling or sampling: Collective dynamics for electrical flow estimation. In *Proc. of the 17th Intl. Conf. on Autonomous Agents and MultiAgent Systems*, AAMAS '18, pages 1576–1584, 2018.
- [8] A. Broder. Generating random spanning trees. In *Proc. of the 30th Annual Symp. on Foundations of Computer Science*, SFCS '89, pages 442–447. IEEE Computer Society, 1989.
- [9] P. Christiano, J. A. Kelner, A. Mądry, D. A. Spielman, and S. H. Teng. Electrical flows, laplacian systems, and faster approximation of maximum flow in undirected graphs. In *Proc. of the 43rd annual ACM Symp. on Theory of Computing*, STOC '11, pages 273–282. ACM, 2011.
- [10] D. Durfee, R. Kyng, J. Peebles, A. B. Rao, and S. Sachdeva. Sampling random spanning trees faster than matrix multiplication. In *Proc. of the 49th Annual ACM Symp. on Theory of Computing*, STOC '17, pages 730–742. ACM, 2017.
- [11] M. Ghaffari and B. Haeupler. Brief announcement: Near-optimal bfs-tree construction in radio networks. In *Proc. of the 2014 ACM Symp. on Principles of Distributed Computing*, PODC '14, 2014.
- [12] I. A. Gillani, A. Bagchi, and P. Vyavahare. Decentralized random walk-based data collection in networks. arXiv:1701.05296 [cs.NI]., 2017.
- [13] J. A. Kelner and A. Mądry. Faster generation of random spanning trees. In *Proc. of the 50th Annual IEEE Symp. on Foundations of Computer Science*, FOCS '09, pages 13–21. IEEE, 2009.
- [14] T. Leighton and S. Rao. Multicommodity max-flow min-cut theorems and their use in designing approximation algorithms. *J. ACM*, 46(6):787–832, November 1999.
- [15] D. A. Levin, Y. Peres, and E. L. Wilmer. *Markov chains and mixing times*. American Mathematical Soc., 2009.

- [16] R. B. Lund and R. L. Tweedie. Geometric convergence rates for stochastically ordered markov chains. *Mathematics of operations research*, 21(1):182–194, 1996.
- [17] G. L. Miller, R. Peng, and S. C. Xu. Parallel graph decompositions using random shifts. In *Proceedings of the Twenty-fifth Annual ACM Symposium on Parallelism in Algorithms and Architectures*, SPAA '13, pages 196–203. ACM, 2013.
- [18] A. Mađry. Computing maximum flow with augmenting electrical flows. In *Proc. of the 57th Annual IEEE Symp. on Foundations of Computer Science*, FOCS '16, pages 593–602. IEEE, 2016.
- [19] A. Mađry, D. Straszak, and J. Tarnawski. Fast generation of random spanning trees and the effective resistance metric. In *Proc. of the 26th Annual ACM-SIAM Symp. on Discrete Algorithms*, SODA '15, pages 2019–2036. SIAM, 2015.
- [20] D Peleg. Distributed computing: A locality-sensitive approach. siam, 2000. *Monographs in Discrete Mathematics and Applications*, 2000.
- [21] R. Peng. Approximate undirected maximum flows in $o(\text{mpolylog}(n))$ time. In *Proc. of the 27th Annual ACM-SIAM Symp. on Discrete Algorithms*, SODA '16, pages 1862–1867. SIAM, 2016.
- [22] R. Peng and D. A. Spielman. An efficient parallel solver for sdd linear systems. In *Proc. of the 46th annual ACM Symp. on Theory of computing*, STOC '14, pages 333–342. ACM, 2014.
- [23] P. Rebeschini and S. Tatikonda. A new approach to laplacian solvers and flow problems. arXiv:1611.07138 [math.OC], 2016.
- [24] A. Schild. An almost-linear time algorithm for uniform random spanning tree generation. In *Proc. of the 50th annual ACM Symp. on Theory of computing*, STOC '18, pages 214–227. ACM, 2018.
- [25] D. A. Spielman and N. Srivastava. Graph sparsification by effective resistances. *SIAM J. Computing*, 40(6):1913–1926, 2011.
- [26] D. A. Spielman and S. H. Teng. Nearly-linear time algorithms for graph partitioning, graph sparsification, and solving linear systems. In *Proc. of the 36th Annual ACM Symp. on Theory of Computing*, STOC '04. ACM, 2004.
- [27] N. K. Vishnoi. $Lx = b$ laplacian solvers and their algorithmic applications. *Foundations and Trends ® in Theoretical Computer Science*, 8(1–2):1–141, 2013.
- [28] A. Zouzias and N. M. Freris. Randomized gossip algorithms for solving laplacian systems. In *European Control Conf.*, ECC '15, pages 1920–1925. IEEE, 2015.