Near-Optimal Sublinear Time Bounds for Distributed Random Walks

Atish Das Sarma * Danupon Nanongkai * Gopal Pandurangan † Prasad Tetali ‡

Abstract

We focus on the problem of performing random walks efficiently in a distributed network. Given bandwidth constraints, the goal is to minimize the number of rounds required to obtain a random walk sample. Despite the widespread use of random walks in distributed computing theory and practice for long, most algorithms that compute a random walk sample of length ℓ always do so naively, i.e., in $O(\ell)$ rounds. Recently, a significantly faster sublinear time distributed algorithm was presented that ran in $\tilde{O}(\ell^{2/3}D^{1/3})$ rounds [1] where D is the diameter of the network [6]. This was the first result to improve beyond linear time (in ℓ) despite the sequential nature of random walks. This work further conjectured that a running time of $\tilde{O}(\sqrt{\ell D})$ is possible and that this is essentially optimal.

In this paper, we resolve these conjectures and show almost tight bounds on the time complexity of distributed random walks. We present a fast distributed algorithm for performing random walks. Our algorithm performs a random walk of length ℓ in $\tilde{O}(\sqrt{\ell D})$ rounds on an undirected network, where D is the diameter of the network. We then show that there is a fundamental difficulty in improving the dependence on ℓ any further by proving a lower bound of $\Omega(\sqrt{\frac{\ell}{\log \ell}} + D)$ rounds for performing a walk of length ℓ. This shows that our algorithm is optimal (up to polylogarithmic factors and the dependence on D).

We further extend our algorithms to perform k independent random walks in roughly $\tilde{O}(\sqrt{k\ell D} + k)$ rounds. Our techniques can be useful in speeding up distributed algorithms for a variety of applications that use random walks as a subroutine. We illustrate one such application involving the decentralized computation of mixing time, a key global parameter of the underlying network. Our algorithms are fully decentralized and can serve as building blocks in the design of topologically-aware networks.

Keywords: Random walks, Random sampling, Decentralized computation, Distributed algorithms.

*College of Computing, Georgia Institute of Technology, Atlanta, GA 30332, USA. E-mail: atish@cc.gatech.edu, danupon@cc.gatech.edu

†Division of Mathematical Sciences, Nanyang Technological University, Singapore 637371 and Department of Computer Science, Brown University, Providence, RI 02912. E-mail: gopalpandurangan@gmail.com. Supported in part by NSF grant CCF-0830476.

‡School of Mathematics and School of Computer Science, Georgia Institute of Technology Atlanta, GA 30332, USA. E-mail: tetali@math.gatech.edu.

[1] Throughout this paper, $\tilde{O}$ hides polylogarithmic factors in the number of nodes in the network

1 Introduction

Random walks play a central role in computer science, spanning a wide range of areas in both theory and practice. The focus of this paper is random walks in networks, in particular, decentralized algorithms for performing random walks in arbitrary networks. Random walks are used as an integral subroutine in a wide variety of network applications ranging from token management and load balancing to search, routing, information propagation and gathering, network topology construction and building random spanning trees (e.g., see [6] and the references therein). Random walks are also very useful in providing uniform and efficient solutions to distributed control of dynamic networks [3, 24]. Random walks are local and lightweight and require little index or state maintenance which make them especially attractive to self-organizing dynamic networks such as Internet overlay and ad hoc wireless networks.

A key purpose of random walks in many of these network applications is to perform node sampling. While the sampling requirements in different applications vary, whenever a true sample is required from a random walk of certain steps, all applications perform the walks naively — by simply passing a token from one node to its neighbor: thus to perform a random walk of length ℓ takes time linear in ℓ.

In this paper, we present fast, almost optimal, sublinear time distributed random walk sampling algorithms that are significantly faster than the existing ones. We then present matching lower bounds on the running time of such algorithms and demonstrate that there are fundamental limitations on the speedup that one can achieve. Finally, we apply our random walk algorithms to devise efficient decentralized algorithms for computing key global metrics of the underlying network such as mixing time, spectral gap, and conductance. Such algorithms can be useful building blocks in the design of *topologically (self-)aware* networks, i.e., networks that can monitor and regulate themselves in a decentralized fashion. For example, efficiently computing the mixing time or the spectral gap, allows the network to monitor connectivity and expansion properties of the network.

1.1 Distributed Computing Model

Consider an undirected, unweighted, connected n-node graph $G = (V, E)$. Suppose that every node (vertex) hosts a processor with unbounded computational power, but with limited initial knowledge. Specifically, assume that each node is associated with a distinct identity number from the set $\{1, 2, ..., n\}$. At the beginning of the computation, each node v accepts as input its own identity number and the identity numbers of its neighbors in G. The node may also accept some additional inputs as specified by the problem at hand. The nodes are allowed to communicate through the edges of the graph G. The communication is synchronous, and occurs in discrete pulses, called *rounds*. In particular, all the nodes wake up simultaneously at the beginning of round 1, and from this point on the nodes always know the number of the current round. In each round each node v is allowed to send an arbitrary message of size $O(\log n)$ through each edge $e = (v, u)$ that is adjacent to v, and the message will arrive to u at the end of the current round. This is a standard model of distributed computation known as the *CONGEST model* [20] and has been attracting a lot of research attention during last two decades (e.g., see [20] and the references therein).

There are several measures of efficiency of distributed algorithms, but we will concentrate on one of them, specifically, *the running time*, that is, the number of rounds of distributed communication. (Note that the computation that is performed by the nodes locally is "free", i.e., it does not affect the number of rounds.) Many fundamental network problems such as minimum spanning tree, shortest paths, etc. have been addressed in this model (e.g., see [16, 20, 19]). In particular, there has been much research into designing very fast distributed approximation algorithms (that are even faster at the cost of producing sub-optimal solutions) for many of these problems (see e.g., [8, 7, 14, 13]). Such algorithms are especially useful for large-scale resource-constrained and dynamic networks where running time is crucial.

1.2 Problem Statement and Related Work

The basic problem we address is the following. We are given an arbitrary undirected, unweighted, and connected n–node network $G = (V, E)$ and a node $s \in V$. The goal is to devise a distributed algorithm such

that, in the end, s outputs the ID of a node v which is randomly picked according to the probability that it is the destination of a random walk of length ℓ starting at s. Throughout this paper, we assume the standard random walk: in each step, an edge is taken from the current node x with probability proportional to $1/d(x)$ where $d(x)$ is the degree of x. Our goal is to output a true random sample from the ℓ-walk distribution starting from s.

For clarity, observe that the following naive algorithm solves the above problem in $O(\ell)$ rounds: The walk of length ℓ is performed by sending a token for ℓ steps, picking a random neighbor with each step. Then, the destination node v of this walk sends its ID back (along the same path) to the source for output. Our goal is to perform such sampling with significantly less number of rounds.

This problem was proposed in [6] under the name *Computing One Random Walk where Source Outputs Destination (1-RW-SoD)* (for short, this problem will be simply called *Single Random Walk* in this paper), wherein the first sublinear time distributed algorithm was provided, requiring $\tilde{O}(\ell^{2/3}D^{1/3})$ rounds ($\tilde{O}$ hides $\text{polylog}(n)$ factors); this improves over the naive $O(\ell)$ algorithm when the walk is long compared to the diameter (i.e., $\ell = \Omega(D \operatorname{polylog} n)$ where D is the diameter of the network). This was the first result to break past the inherent sequential nature of random walks and beat the naive ℓ round approach, despite the fact that random walks have been used in distributed networks for long and in a wide variety of applications. It was further conjectured in [6] that the true number of rounds for this problem is $\tilde{O}(\sqrt{\ell D})$.

The high-level idea used in the $\tilde{O}(\ell^{2/3}D^{1/3})$-round algorithm in [6] is to "prepare" a few short walks in the beginning (executed in parallel) and then carefully stitch these walks together later as necessary. The same general approach was introduced in [5] to find random walks in data streams with the main motivation of finding PageRank. However, the two models have very different constraints and motivations and hence the subsequent techniques used in [6] and [5] are very different. Our algorithms in this paper use the same general approach as [6] but exploit certain key properties of random walks to design even faster sublinear time algorithms.

Recently, Sami and Twigg [22] consider lower bounds on the communication complexity of computing stationary distribution of random walks in a network. Although, their problem is related to our problem, the lower bounds obtained do not imply anything in our setting. Other recent works involving multiple random walks in different settings include Alon et. al. [1], and Cooper et al. [4].

1.3 Our Results

- **Upper Bound:** We present a sublinear, almost time-optimal, distributed algorithm for the single random walk problem in arbitrary networks that runs in time $\tilde{O}(\sqrt{\ell D})$, where ℓ is the length of the walk (cf. Section 2). This is a significant improvement over the naive ℓ-round algorithm for $\ell = \Omega(D)$ as well as over the previous best running time of $\tilde{O}(\ell^{2/3}D^{1/3})$ [6]. The dependence on ℓ is reduced from $\ell^{2/3}$ to $\ell^{1/2}$. Many real-world communication networks (e.g., ad hoc networks and peer-to-peer networks) have relatively small diameter, and random walks of length at least the diameter are usually performed, i.e., $\ell >> D$. It should be noted that if the network is rapidly mixing/expanding which is sometimes the case in practice, then sampling from walks of length $\ell >> D$ is close to sampling from the steady state (degree) distribution; this can be done in $O(D)$ rounds. However, such an approach fails when ℓ is smaller than the mixing time.

  One of the key ingredients in obtaining our upper bound is proving a bound on the number of times any node is visited in an ℓ-length walk, for any arbitrary length ℓ. We show that w.h.p. any node x is visited at most $\tilde{O}(d(x)\sqrt{\ell})$ times, in an ℓ-length walk from any starting node ($d(x)$ is the degree of x). We then show that if only certain ℓ/λ special points of the walk (called as *connector points*) are observed, then any node is observed only $\tilde{O}(d(x)\sqrt{\ell}/\lambda)$ times. The algorithm starts with all nodes performing short walks (of length uniformly random in the range λ to 2λ for appropriately chosen λ) efficiently simultaneously; here the randomly chosen lengths play a crucial role in arguing about a suitable spread of the connector points. Subsequently, the algorithm begins at the source and carefully stitches these walks together till ℓ steps are completed.

We also extend to give algorithms for computing k random walks (from any k sources —not necessarily distinct) in $\tilde{O}\left(\min(\sqrt{k\ell D}+k, \sqrt{k\ell}+k+\ell)\right)$ rounds. Computing k random walks is useful in many applications such as the one we present below on decentralized computation of mixing time and related parameters. While the main requirement of our algorithms is to just obtain the random walk samples (i.e. the end point of the ℓ step walk), our algorithms can regenerate the entire walks such that each node knows its position(s) among the ℓ steps.

- **Lower Bound:** We establish an almost matching lower bound on the running time of a class of distributed algorithms for performing a random walk. We show that any such algorithm needs at least $\Omega(\sqrt{\frac{\ell}{\log \ell}}+D)$ rounds to perform a walk of length ℓ; notice that this lower bound is nontrivial even in graphs of small ($D = O(\log n)$) diameter (cf. Section 3).

  Our technique involves showing the same non-trivial lower bound for a natural class of distributed algorithms for a problem that we call *path verification*. This simpler problem appears quite fundamental and can have other applications. Informally, given a graph G and a sequence of ℓ vertices in the graph, the problem is for some (source) node in the graph to verify that the sequence forms a path. One main idea in this proof is to show that independent nodes may be able to verify short *local* paths; however, to be able to *merge* these together and verify an ℓ-length path would require exchanging several messages. The trade-off is between the lengths of the local paths that are verified and the number of such local paths that need to be combined. Locally verified paths can be exchanged in one round, and messages can be exchanged at all nodes. Despite this, we show that the bandwidth restriction necessitates a large number of rounds even if the diameter is small. We then show a reduction to the random walk problem.

  Similar non-trivial matching lower bounds on running time are known only for a few important problems in distributed computing, notably the minimum spanning tree problem (e.g., see [21, 9]). Peleg and Rabinovich [21] showed that $\tilde{\Omega}(\sqrt{n})$ time is required for constructing an MST even on graphs of small diameter (for any $D = \Omega(\log n)$) and this is essentially the best possible [15].

- **Applications:** Our faster distributed random walk algorithm can be used in speeding up distributed applications that use random walks as a subroutine. Such applications include distributed construction of expander graphs, checking whether a graph is an expander, construction of random spanning trees, and random-walk based search (we refer to [6] for details). Here we present a concrete application of our algorithm (cf. Section 4) by showing that it can be useful in estimating mixing time, conductance and spectral gap of the network. In particular, we show that given a starting point x, the mixing time with respect to x, called τ^x_{mix}, can be estimated in $\tilde{O}(n^{1/2}+n^{1/4}\sqrt{D\tau^x_{mix}})$ rounds. This gives an alternative algorithm to the only previously known approach by Kempe and McSherry [12] that can be used to estimate τ^x_{mix} in $\tilde{O}(\tau^x_{mix})$ rounds.[2] To compare, we note that when $\tau^x_{mix} = \omega(n^{1/2})$ the present algorithm is faster (assuming D is not too large).

  The work of [10] discusses spectral algorithms for enhancing the topology awareness, e.g., by identifying and assigning weights to critical links. However, the algorithms are centralized, and it is mentioned that obtaining efficient decentralized algorithms is a major open problem. Our algorithms are fully decentralized and based on performing random walks, and so more amenable to dynamic and self-organizing networks.

2 Upper Bound: A Sublinear Time Distributed Random Walk Algorithm

2.1 Description of the Algorithm

We first describe the $\tilde{O}(\ell^{2/3}D^{1/3})$-round algorithm in [6] and then highlight the changes in our current algorithm. The current algorithm uses several new ideas that are crucial in obtaining the new bound.

[2]Note that [12] in fact do more and give a decentralized algorithm for computing the top k eigenvectors of a weighted adjacency matrix that runs in $O(\tau_{mix}\log^2 n)$ rounds if two adjacent nodes are allowed to exchange $O(k^3)$ messages per round, where τ_{mix} is the mixing time and n is the size of the network.

The high-level idea is to perform "many" short random walks in parallel and later stitch them together as needed (see Figure 2 in Appendix). In the first phase of the algorithm SINGLE-RANDOM-WALK, each node performs η independent random walks of length λ. (Only the destination of each of these walks is aware of its source, but the sources do not know destinations right away.) It is shown that this takes $\tilde{O}(\eta\lambda)$ rounds with high probability. Subsequently, the source node that requires a walk of length ℓ extends a walk of length λ by "stitching" walks. If the end point of the first λ length walk is u, one of u's λ length walks is used to extend. When at u, one of its λ-length walk destinations are sampled uniformly (to preserve randomness) using SAMPLE-DESTINATION in $O(D)$ rounds. (We call such u and other nodes at the stitching points as *connectors* — cf. Algorithm 1.) Each stitch takes $O(D)$ rounds (via the shortest path). This process is extended as long as unused λ-length walks are available from visited nodes. If the walk reaches a node v where all η walks have been used up (which is a key difficulty), then GET-MORE-WALKS is invoked. GET-MORE-WALKS performs η more walks of length λ from v, and this can be done in $\tilde{O}(\lambda)$ rounds. The number of times GET-MORE-WALKS is invoked can be bounded by $\frac{\ell}{\eta\lambda}$ in the worst case by an amortization argument. The overall bound on the algorithm is $O(\eta\lambda + \ell D/\lambda + \frac{\ell}{\eta})$. The bound of $\tilde{O}(\ell^{2/3}D^{1/3})$ follows from appropriate choice of parameters η and λ.

The current algorithm uses two crucial ideas to improve the running time. The first idea is to bound the number of times any node is visited in a random walk of length ℓ (in other words, the number of times GET-MORE-WALKS is invoked). Instead of the worst case analysis in [6], the new bound is obtained by bounding the number of times any node is visited (with high probability) in a random walk of length ℓ on an undirected unweighted graph. The number of visits to a node beyond the mixing time can be bounded using its stationary probability distribution. However, we need a bound on the visits to a node for any ℓ-length walk starting from the first step. We show a somewhat surprising bound that applies to an ℓ-length random walk on any arbitrary (undirected) graph: *no node x is visited more than $\tilde{O}(d(x)\sqrt{\ell})$ times*, in an ℓ-length walk from any starting node ($d(x)$ is the degree of x) (cf. Lemma 2.8). Note that this bound does not depend on any other parameter of the graph, just on the (local) degree of the node and the length of the walk. This bound is tight in general (e.g., consider a line and a walk of length n).

The above bound is not enough to get the desired running time, as it does not say anything about the distribution of connectors when we chop the length ℓ walk into ℓ/λ pieces. We have to bound the number of visits to a node as a connector in order to bound the number of times GET-MORE-WALKS is invoked. To overcome this we use a second idea: Instead of nodes performing walks of length λ, each such walk i is of length $\lambda + r_i$ where r_i is a random number in the range $[0, \lambda - 1]$. Notice that the random numbers are independent for each walk. We show the following "uniformity lemma": if the short walks are now of a random length in the range of $[\lambda, 2\lambda - 1]$, then if a node u is visited at most N_u times in an ℓ step walk, then the node is visited at most $\tilde{O}(N_u/\lambda)$ times as an endpoint of a short walk (cf. Lemma 2.12). This modification to SINGLE-RANDOM-WALK allows us to bound the number of visits to each node (cf. Lemma 2.12).

The change of the short walk length above leads to two modifications in Phase 1 of SINGLE-RANDOM-WALK and GET-MORE-WALKS. In Phase 1, generating η walks of different lengths from each node is straightforward: Each node simply sends η tokens containing the source ID and the desired length. The nodes keep forwarding these tokens with decreased desired walk length until the desired length becomes zero. The modification of GET-MORE-WALKS is tricker. To avoid congestion, we use the idea of *reservoir sampling* [23]. In particular, we add the following process at the end of the GET-MORE-WALKS algorithm in [6]:

for $i = 0$ to $\lambda - 1$ **do**

For each message, independently with probability $\frac{1}{\lambda - i}$, stop sending the message further and save the ID of the source node (in this event, the node with the message is the destination). For messages M that are not stopped, each node picks a neighbor correspondingly and sends the messages forward as before.

end for

The reason it needs to be done this way is that if we first sampled the walk length r, independently for each walk, in the range $[0, \lambda - 1]$ and then extended each walk accordingly, the algorithm would need to pass r independently for each walk. This will cause congestion along the edges; no congestion occurs in the mentioned algorithm as only the *count* of the number of walks along an edge are passed to the node across the edge. Therefore, we need to decide when to stop on the fly using reservoir sampling.

We also have to make another modification in Phase 1 due to the new bound on the number of visits. Recall that, in this phase, each node prepares η walks of length λ. However, since the new bound of visits of each node x is proportional to its degree $d(x)$ (see Lemma 2.8), we make each node prepare $\eta d(x)$ walks instead. We show that Phase 1 uses $\tilde{O}(\eta\lambda)$ rounds, instead of $\tilde{O}(\frac{\lambda\eta}{\delta})$ rounds where δ is the minimum degree in the graph (cf. Lemma 2.4).

To summarize, the main algorithm for performing a single random walk is SINGLE-RANDOM-WALK. This algorithm, in turn, uses GET-MORE-WALKS and SAMPLE-DESTINATION. The key modification is that, instead of creating short walks of length λ each, we create short walks where each walk has length in range $[\lambda, 2\lambda - 1]$. To do this, we modify the Phase 1 of SINGLE-RANDOM-WALK and GET-MORE-WALKS. We now present the pseudocodes of these algorithms.

2.2 Algorithm descriptions

The main algorithm for performing a single random walk is described in SINGLE-RANDOM-WALK (cf. Algorithm 1). This algorithm, in turn, uses GET-MORE-WALKS (cf. 2 and SAMPLE-DESTINATION (cf. 3).

Notice that in Line 9 in Algorithm 2, the walks of length λ are extended further to walks of length $\lambda + r$ where r is a random number in the range $[0, \lambda - 1]$. We do this by extending the λ-length walks further, and probabilistically stopping each walk in each of the next i steps (for $0 \leq i \leq \lambda - 1$) with probability $\frac{1}{\lambda - i}$. The reason it needs to be done this way is because if we first sampled r, independently for each walk, in the range $[0, \lambda - 1]$ and then extended each walk accordingly, the algorithm would need to pass r independently for each walk. This will cause congestion along the edges; no congestion occurs in the mentioned algorithm as only the *count* of the number of walks along an edge are passed to the node across the edge.

2.3 Analysis

We now state four lemmas which are similar to the Lemma 2.2-2.6 in [6]. Since our algorithms are modifications of [6], we present the proofs for completeness.

Lemma 2.1. *Phase 1 finishes in $O(\lambda\eta \log n)$ rounds with high probability.*

Proof. This proof is a slight modification of the proof of Lemma 2.2 in [6], where it is shown that each node can perform η walks of length λ together in $O(\lambda\eta \log n)$ rounds with high probability. We extend this to the following statement.

Each node v can in fact perform $\eta \deg(v)$ of length 2λ and still finish in $O(\lambda\eta \log n)$ rounds.

The desired claim will follow immediately because each node v performs $\eta \deg(v)$ of length *at most* λ in Phase 1.

Consider the case when each node v creates $\eta \deg(v) \geq \eta$ messages. For each message M, any $j = 1, 2, ..., \lambda$, and any edge e, we define $X_M^j(e)$ to be a random variable having value 1 if M is sent through e in the j^{th} iteration (i.e., when the counter on M has value $j - 1$). Let $X^j(e) = \sum_{M:\text{message}} X_M^j(e)$. We compute the expected number of messages that go through an edge, see claim below.

Claim 2.2. *For any edge e and any j, $\mathbb{E}[X^j(e)] = 2\eta$.*

Proof. Assume that each node v starts with $\eta \deg(v)$ messages. Each message takes a random walk. We prove that after any given number of steps j, the expected number of messages at node v is still $\eta \deg(v)$.

Algorithm 1 SINGLE-RANDOM-WALK(s, ℓ)

Input: Starting node s, and desired walk length ℓ.
Output: Destination node of the walk outputs the ID of s.

Phase 1: (Each node v performs $\eta_v = \eta \deg(v)$) random walks of length $\lambda + r_i$ where r_i (for each $1 \leq i \leq \eta$) is chosen independently at random in the range $[0, \lambda - 1]$.)

1: Let $r_{max} = \max_{1 \leq i \leq \eta} r_i$, the random numbers chosen independently for each of the η_x walks.
2: Each node x constructs η_x messages containing its ID and in addition, the i-th message contains the desired walk length of $\lambda + r_i$.
3: **for** $i = 1$ to $\lambda + r_{max}$ **do**
4: This is the i-th iteration. Each node v does the following: Consider each message M held by v and received in the $(i-1)$-th iteration (having current counter $i - 1$). If the message M's desired walk length is at most i, then v stored the ID of the source (v is the desired destination). Else, v picks a neighbor u uniformly at random and forward M to u after incrementing its counter.
{Note that any iteration could require more than 1 round.}
5: **end for**

Phase 2: (Stitch $\Theta(\ell/\lambda)$ walks, each of length in $[\lambda, 2\lambda - 1]$)

1: The source node s creates a message called "token" which contains the ID of s
2: The algorithm generates a set of *connectors*, denoted by C, as follows.
3: Initialize $C = \{s\}$
4: **while** Length of walk completed is at most $\ell - 2\lambda$ **do**
5: Let v be the node that is currently holding the token.
6: v calls SAMPLE-DESTINATION(v) and let v' be the returned value (which is a destination of an unused random walk starting at v of length between λ and $2\lambda - 1$.)
7: **if** v' = NULL (all walks from v have already been used up) **then**
8: v calls GET-MORE-WALKS(v, λ) (Perform $\Theta(l/\lambda)$ walks of length λ starting at v)
9: v calls SAMPLE-DESTINATION(v) and let v' be the returned value
10: **end if**
11: v sends the token to v'
12: $C = C \cup \{v\}$
13: **end while**
14: Walk naively until ℓ steps are completed (this is at most another 2λ steps)
15: A node holding the token outputs the ID of s

Algorithm 2 GET-MORE-WALKS(v, λ)

(Starting from node v, perform $\lfloor \ell/\lambda \rfloor$ number of random walks, each of length $\lambda + r_i$ where r_i is chosen uniformly at random in the range $[0, \lambda - 1]$ for the i-th walk.)

1: The node v constructs $\lfloor \ell/\lambda \rfloor$ (identical) messages containing its ID.
2: **for** $i = 1$ to λ **do**
3: Each node u does the following:
4: - For each message M held by u, pick a neighbor z uniformly at random as a receiver of M.
5: - For each neighbor z of u, send ID of v and the number of messages that z is picked as a receiver, denoted by $c(u, v)$.
6: - For each neighbor z of u, upon receiving ID of v and $c(u, v)$, constructs $c(u, v)$ messages, each contains the ID of v.
7: **end for**
{Each walk has now completed λ steps. These walks are now extended probabilistically further by r steps where each r is independent and uniform in the range $[0, \lambda - 1]$.}
8: **for** $i = 0$ to $\lambda - 1$ **do**
9: For each message, independently with probability $\frac{1}{\lambda - i}$, stop sending the message further and save the ID of the source node (in this event, the node with the message is the destination). For messages M that are not stopped, each node picks a neighbor correspondingly and sends the messages forward as before.
10: **end for**
11: At the end, each destination knows the source ID as well as the length of the corresponding walk.

Consider the random walk's probability transition matrix, call it A. In this case $Au = u$ for the vector u having value $\frac{\deg(v)}{2m}$ where m is the number of edges in the graph (since this u is the stationary distribution of an undirected unweighted graph). Now the number of messages we started with at any node i is proportional to its stationary distribution, therefore, in expectation, the number of messages at any node remains the same.

To calculate $\mathbb{E}[X^j(e)]$, notice that edge e will receive messages from its two end points, say x and y. The number of messages it receives from node x in expectation is exactly the number of messages at x divided by $\deg(x)$. The claim follows. □

By Chernoff's bound (e.g., in [18, Theorem 4.4.]), for any edge e and any j,

$$\mathbb{P}[X^j(e) \geq 4\eta \log n] \leq 2^{-4 \log n} = n^{-4}.$$

It follows that the probability that there exists an edge e and an integer $1 \leq j \leq \lambda$ such that $X^j(e) \geq 4\eta \log n$ is at most $|E(G)|\lambda n^{-4} \leq \frac{1}{n}$ since $|E(G)| \leq n^2$ and $\lambda \leq \ell \leq n$ (by the way we define λ).

Now suppose that $X^j(e) \leq 4\eta \log n$ for every edge e and every integer $j \leq \lambda$. This implies that we can extend all walks of length i to length $i + 1$ in $4\eta \log n$ rounds. Therefore, we obtain walks of length λ in $4\lambda\eta \log n$ rounds as claimed. □

Lemma 2.3. *For any v,* GET-MORE-WALKS(v, η, λ) *always finishes within $O(\lambda)$ rounds.*

Proof. The argument is exactly the same as the proof of Lemma 2.4 in [6]. That is, there is no congestion. We only consider longer walks (length at most $2\lambda - 1$) this time. The detail of the proof is as follows.

Consider any node v during the execution of the algorithm. If it contains x copies of the source ID, for some x, it has to pick x of its neighbors at random, and pass the source ID to each of these x neighbors. Although it might pass these messages to less than x neighbors, it sends only the source ID and a *count* to each neighbor, where the count represents the number of copies of source ID it wishes to send to such

Algorithm 3 SAMPLE-DESTINATION(v)

Input: Starting node v.

Output: A node sampled from among the stored walks (of length in $[\lambda, 2\lambda - 1]$) from v.

Sweep 1: (Perform BFS tree)

1: Construct a Breadth-First-Search (BFS) tree rooted at v. While constructing, every node stores its parent's ID. Denote such tree by T.

Sweep 2: (Tokens travel up the tree, sample as you go)

1: We divide T naturally into levels 0 through D (where nodes in level D are leaf nodes and the root node s is in level 0).
2: Tokens are held by nodes as a result of doing walks of length between λ and $2\lambda-1$ from v (which is done in either Phase 1 or GET-MORE-WALKS (cf. Algorithm 2)) A node could have more than one token.
3: Every node u that holds token(s) picks one token, denoted by d_0, uniformly at random and lets c_0 denote the number of tokens it has.
4: **for** $i = D$ down to 0 **do**
5: Every node u in level i that either receives token(s) from children or possesses token(s) itself do the following.
6: Let u have tokens $d_0, d_1, d_2, \ldots, d_q$, with counts $c_0, c_1, c_2, \ldots, c_q$ (including its own tokens). The node v samples one of d_0 through d_q, with probabilities proportional to the respective counts. That is, for any $1 \leq j \leq q$, d_j is sampled with probability $\frac{c_j}{c_0+c_1+\ldots+c_q}$.
7: The sampled token is sent to the parent node (unless already at root), along with a count of $c_0 + c_1 + \ldots + c_q$ (the count represents the number of tokens from which this token has been sampled).
8: **end for**
9: The root output the ID of the owner of the final sampled token. Denote such node by u_d.

Sweep 3: (Go and delete the sampled destination)

1: v sends a message to u_d (e.g., via broadcasting). u_d deletes one token of v it is holding (so that this random walk of length λ is not reused/re-stitched).

neighbor. Note that there is only one source ID as one node calls GET-MORE-WALKS at a time. Therefore, there is no congestion and thus the algorithm terminates in $O(\lambda)$ rounds. □

Lemma 2.4. SAMPLE-DESTINATION *always finishes within* $O(D)$ *rounds.*

Proof. This proof is exactly the same as the proof of Lemma 2.5 in [6].

Constructing a BFS tree clearly takes only $O(D)$ rounds. In the second phase where the algorithm wishes to *sample* one of many tokens (having its ID) spread across the graph. The sampling is done while retracing the BFS tree starting from leaf nodes, eventually reaching the root. The main observation is that when a node receives multiple samples from its children, it only sends one of them to its parent. Therefore, there is no congestion. The total number of rounds required is therefore the number of levels in the BFS tree, $O(D)$. The third phase of the algorithm can be done by broadcasting (using a BFS tree) which needs $O(D)$ rounds. □

Lemma 2.5. *Algorithm* SAMPLE-DESTINATION*(v) (cf. Algorithm 3) returns a destination from a random walk whose length is uniform in the range* $[\lambda, 2\lambda - 1]$.

Proof of Lemma 2.5. The claim follows from the correctness of SAMPLE-DESTINATION that the algorithm samples a walk uniformly at random and the fact that the length of each walk is uniformly sampled from the range $[\lambda, 2\lambda - 1]$. The first part is proved in Lemma 2.6 in Das Sarma et al. [6] and included below for completeness. We now prove the second part.

To show that each walk length is uniformly sampled from the range $[\lambda, 2\lambda - 1]$, note that each walk can be created in two ways.

1. It is created in Phase 1. In this case, since we pick the length of each walk uniformly from the length $[\lambda, 2\lambda - 1]$, the claim clearly holds.

2. It is created by GET-MORE-WALK. In this case, the claim holds by the technique of *reservoir* sampling: Observe that after the λ^{th} step of the walk is completed, we stop extending each walk at any length between λ and $2\lambda - 1$ uniformly. To see this, observe that we stop at length λ with probability $1/\lambda$. If the walk does not stop, it will stop at length $\lambda + 1$ with probability $\frac{1}{\lambda-1}$. This means that the walk will stop at length $\lambda + 1$ with probability $\frac{\lambda-1}{\lambda} \times \frac{1}{\lambda-1} = \frac{1}{\lambda-1}$. Similarly, it can be argue that the walk will stop at length i for any $i \in [\lambda, 2\lambda - 1]$ with probability $\frac{1}{\lambda}$.

We now show the proof of Lemma 2.6 (with slight modification) in Das Sarma et al. for completeness

Lemma 2.6 (Lemma 2.6 in [6])**.** *Algorithm* SAMPLE-DESTINATION*(v) (cf. Algorithm 3), for any node v, samples a destination of a walk starting at v uniformly at random.*

Proof. Assume that before this algorithm starts, there are t (without loss of generality, let $t > 0$) "tokens" containing ID of v stored in some nodes in the network. The goal is to show that SAMPLE-DESTINATION brings one of these tokens to v with uniform probability. For any node u, let T_u be the subtree rooted at u and let S_u be the set of tokens in T_u. (Therefore, $T_v = T$ and $|S_v| = t$.)

We claim that any node u returns a destination to its parent with uniform probability (i.e., for any tokens $x \in S_u$, $Pr[u \text{ returns } x]$ is $1/|S_u|$ (if $|S_u| > 0$)). We prove this by induction on the height of the tree. This claim clearly holds for the base case where u is a leaf node. Now, for any non-leaf node u, assume that the claim is true for any of its children. To be precise, suppose that u receives tokens and counts from q children. Assume that it receives tokens $d_1, d_2, ..., d_q$ and counts $c_1, c_2, ..., c_q$ from nodes $u_1, u_2, ..., u_q$, respectively. (Also recall that d_0 is the sample of its own tokens (if exists) and c_0 is the number of its own tokens.) By induction, d_j is sent from u_j to u with probability $1/|S_{u_j}|$, for any $1 \leq j \leq q$. Moreover, $c_j = |S_{u_j}|$ for any j. Therefore, any token d_j will be picked with probability $\frac{1}{|S_{u_j}|} \times \frac{c_j}{c_0+c_1+...c_q} = \frac{1}{S_u}$ as claimed.

The lemma follows by applying the claim above to v. □

□

The following theorem states the main result of this Section. It states that the algorithm SINGLE-RANDOM-WALK correctly samples a node after a random walk of ℓ steps and the algorithm takes, with high probability, $\tilde{O}\left(\sqrt{\ell D}\right)$ rounds where D is the diameter of the graph.

Theorem 2.7. *For any ℓ, Algorithm* SINGLE-RANDOM-WALK *(cf. Algorithm 1) solves 1-RW-DoS (the Single Random Walk Problem) and, with probability at least $1 - \frac{2}{n}$, finishes in $\tilde{O}\left(\sqrt{\ell D}\right)$ rounds.*

We prove the above theorem using the following lemmas. As mentioned earlier, to bound the number of times GET-MORE-WALKS is invoked, we need a technical result on random walks that bounds the number of times a node will be visited in a ℓ-length random walk. Consider a simple random walk on a connected undirected graph on n vertices. Let $d(x)$ denote the degree of x, and let m denote the number of edges. Let $N_t^x(y)$ denote the number of visits to vertex y by time t, given the walk started at vertex x. Now, consider k walks, each of length ℓ, starting from (not necessary distinct) nodes $x_1, x_2, \ldots, x_k$. We show a key technical lemma that applies to a random walk on any graph: With high probability, no vertex y is visited more than $24d(x)\sqrt{k\ell + 1}\log n + k$ times.

Lemma 2.8. *For any nodes $x_1, x_2, \ldots, x_k$,*

$$\Pr\big(\exists y \; s.t. \; \sum_{i=1}^{k} N_\ell^{x_i}(y) \geq 24d(x)\sqrt{k\ell+1}\log n + k\big) \leq 1/n\,.$$

Proof. We start with the bound of the first and second moment of the number of visits at each node by each walk.

Proposition 2.9. *For any node x, node y and $t \in Z^+$,*

$$\mathrm{E}[N_t^x(y)] \leq 8d(y)\sqrt{t+1}\,, \quad \text{and} \quad \mathrm{E}\Big[\big(N_t^x(y)\big)^2\Big] \leq \mathrm{E}[N_t^x(y)] + 128\; d^2(y)\;(t+1)\,. \tag{1}$$

To prove the above proposition, let P denote the transition probability matrix of such a random walk and let π denote the stationary distribution of the walk, which in this case is simply proportional to the degree of the vertex, and let $\pi_{\min} = \min_x \pi(x)$.

The basic bound we use is the following estimate from Lyons (see Lemma 3.4 and Remark 4 in [17]). Let Q denote the transition probability matrix of a chain with self-loop probablity $\alpha > 0$, and with $c = \min\{\pi(x)Q(x,y) : Q(x,y) > 0\}$. Note that for a random walk on an undirected graph, $c = 2m$. For $k > 0$ a positive integer (denoting time) ,

$$\Big|\frac{Q^k(x,y)}{\pi(y)} - 1\Big| \leq \min\Big\{\frac{1}{\alpha c\sqrt{k+1}}, \frac{1}{2\alpha^2 c^2(k+1)}\Big\}\,. \tag{2}$$

Sometimes, it is more convenient to use the following bound; see Remark 3 in [17].

$$Q^k(x,y) \leq \frac{4\pi(y)}{c\sqrt{k+1}} = \frac{4d(y)}{\sqrt{k+1}}\,. \tag{3}$$

Note that given a simple random walk on a graph G, and a corresponding matrix P, one can always switch to the lazy version $Q = (I+P)/2$, and interpret it as a walk on graph G', obtained by adding self-loops to vertices in G so as to double the degree of each vertex. In the following, with abuse of notation we assume our P is such a lazy version of the original one.

Proof. Let $X_0, X_1, \ldots$ describe the random walk, with X_i denoting the position of the walk at time $i \geq 0$, and let $\mathbf{1}_A$ denote the indicator (0-1) random variable, which takes the value 1 when the event A is true. In the following we also use the subscript x to denote the fact that the probability or expectation is with respect to starting the walk at vertex x. First the expectation.

$$\begin{aligned}
\mathrm{E}[N_t^x(y)] &= \mathrm{E}_x[\sum_{i=0}^{t} \mathbf{1}_{\{X_i=y\}}] = \sum_{i=0}^{t} P^i(x,y) \\
&\leq 4d(y)\sum_{i=0}^{t}\frac{1}{\sqrt{i+1}}, \quad \text{(using the above inequality (3))} \\
&\leq 8d(y)\sqrt{t+1}\,.
\end{aligned}$$

One does not expect Gaussian tails here, since the random variable in question is distributed more like a Geometric one.

Abbreviating $N_t^x(y)$ as $N_t(y)$, we now compute the second moment:

$$\begin{aligned}
\mathrm{E}[N_t^2(y)] &= \mathrm{E}_x\Big[\big(\sum_{i=0}^{t}\mathbf{1}_{\{X_i=y\}}\big)\big(\sum_{j=0}^{t}\mathbf{1}_{\{X_j=y\}}\big)\Big] \\
&= \mathrm{E}_x\Big[\sum_{i=0}^{t}\mathbf{1}_{\{X_i=y\}} + 2\sum_{0\le i<j\le t}\mathbf{1}_{\{X_i=y,\ X_j=y\}}\Big] \\
&= \mathrm{E}[N_t(y)] + 2\sum_{0\le i<j\le t}\Pr(X_i=y,\ X_j=y)\,.
\end{aligned}$$

To bound the second term on the right hand side above, consider for $0 \le i < j$:

$$\begin{aligned}
\Pr(X_i=y,\ X_j=y) &= \Pr(X_i=y)\ \Pr(X_j=y|X_i=y) \\
&= P^i(x,y)\ P^{j-i}(y,y)\,, \quad \text{due to the Markovian property} \\
&\le \frac{4d(y)}{\sqrt{i+1}}\ \frac{4d(y)}{\sqrt{j-i+1}}\,. \quad \text{(using (3))}
\end{aligned}$$

Thus,

$$\begin{aligned}
\sum_{0\le i<j\le t}\Pr(X_i=y,\ X_j=y) &\le \sum_{0\le i\le t}\frac{4d(y)}{\sqrt{i+1}}\sum_{0<j-i\le t-i}\frac{4d(y)}{\sqrt{j-i+1}} \\
&= 16d^2(y)\sum_{0\le i\le t}\frac{1}{\sqrt{i+1}}\sum_{0<k\le t-i}\frac{1}{\sqrt{k+1}} \\
&\le 32d^2(y)\sum_{0\le i\le t}\frac{1}{\sqrt{i+1}}\sqrt{t-i+1} \\
&\le 32d^2(y)\sqrt{t+1}\sum_{0\le i\le t}\frac{1}{\sqrt{i+1}} \\
&\le 64d^2(y)\,(t+1)\,,
\end{aligned}$$

which yields the claimed bound on the second moment in the proposition. □

Using the above proposition, we bound the number of visits of each walk at each node, as follows.

Lemma 2.10. *For any $t \in \mathrm{Z}^+$ and any vertex $y \in G$, the random walk started at x satisfies:*

$$\Pr\big(N_t^x(y) \ge 24\ d(y)\sqrt{t+1}\log n\big) \le \frac{1}{n^2}\,.$$

Proof. First, it follows from the Proposition that

$$\Pr\big(N_t^x(y) \ge 2\cdot 12\ d(y)\sqrt{t+1}\big) \le \frac{1}{4}\,. \tag{4}$$

This is done by using the standard Chebyshev argument that for $B>0$, $\Pr\big(N_t(y)\ge B\big) \le \Pr\big(N_t^2(y)\ge B^2\big) \le \frac{\mathrm{E}\left(N_t^2(y)\right)}{B^2}$.

For any r, let $L_r^x(y)$ be the time that the random walk (started at x) visits y for the r^{th} time. Observe that, for any r, $N_t^x(y)\ge r$ if and only if $L_r^x(y)\le t$. Therefore,

$$\Pr(N_t^x(y)\ge r) = \Pr(L_r^x(y)\le t). \tag{5}$$

Let $r^* = 24\, d(y)\sqrt{t+1}$. By (4) and (5), $\Pr(L^x_{r^*}(y) \le t) \le \frac{1}{4}$. We claim that

$$\Pr(L^x_{r^* \log n}(y) \le t) \le \left(\frac{1}{4}\right)^{\log n} = \frac{1}{n^2}\,. \tag{6}$$

To see this, divide the walk into $\log n$ independent subwalks, each visiting y exactly r^* times. Since the event $L^x_{r^* \log n}(y) \le t$ implies that all subwalks have length at most t, (6) follows. Now, by applying (5) again,

$$\Pr(N^x_t(y) \ge r^* \log n) = \Pr(L^x_{r^* \log n}(y) \le t) \le \frac{1}{n^2}$$

as desired.

□

We now extend the above lemma to bound the number of visits of *all* the walks at each particular node.

Lemma 2.11. *For $\gamma > 0$, and $t \in \mathrm{Z}^+$, and for any vertex $y \in G$, the random walk started at x satisfies:*

$$\Pr\big(\sum_{i=1}^{k} N^{x_i}_t(y) \ge 24\, d(y)\sqrt{kt+1}\log n + k\big) \le \frac{1}{n^2}\,.$$

Proof. First, observe that, for any r,

$$\Pr\big(\sum_{i=1}^{k} N^{x_i}_t(y) \ge r - k\big) \le \Pr[N^y_{kt}(y) \ge r].$$

To see this, we construct a walk W of length kt starting at y in the following way: For each i, denote a walk of length t starting at x_i by W_i. Let τ_i and τ_i' be the first and last time (not later than time t) that W_i visits y. Let W_i' be the subwalk of W_i from time τ_i to τ_i'. We construct a walk W by stitching $W_1', W_2', ..., W_k'$ together and complete the rest of the walk (to reach the length kt) by a normal random walk. It then follows that the number of visits to y by $W_1, W_2, \ldots, W_k$ (excluding the starting step) is at most the number of visits to y by W. The first quantity is $\sum_{i=1}^{k} N^{x_i}_t(y) - k$. (The term '$-k$' comes from the fact that we do not count the first visit to y by each W_i which is the starting step of each W_i'.) The second quantity is $N^y_{kt}(y)$. The observation thus follows.

Therefore,

$$\Pr\big(\sum_{i=1}^{k} N^{x_i}_t(y) \ge 24\, d(y)\sqrt{kt+1}\log n + k\big) \le \Pr\big(N^y_{kt}(y) \ge 24\, d(y)\sqrt{kt+1}\log n\big) \le \frac{1}{n^2}$$

where the last inequality follows from Lemma 2.10. □

The lemma follows immediately from Lemma 2.11 by union bounding over all nodes. □

Lemma 2.8 says that the number of visits to each node can be bounded. However, for each node, we are only interested in the case where it is used as a connector. The lemma below shows that the number of visits as a connector can be bounded as well; i.e., if any node v_i appears t times in the walk, then it is likely to appear roughly t/λ times as connectors. The proof follows.

Lemma 2.12. *For any vertex v, if v appears in the walk at most t times then it appears as a connector node at most $t(\log n)^2/\lambda$ times with probability at least $1 - 1/n^2$.*

Intuitively, this argument is simple, since the connectors are spread out in steps of length approximately λ. However, there might be some *periodicity* that results in the same node being visited multiple times but *exactly* at λ-intervals. This is where we crucially use the fact that the algorithm uses walks of length $\lambda + r$ where r is chosen uniformly at random from $[0, \lambda - 1]$. The proof then goes via constructing another process equivalent to partitioning the ℓ steps in to intervals of λ and then sampling points from each interval. We analyze this by carefully constructing a different process that stochastically dominates the process of a node occurring as a connector at various steps in the ℓ-length walk and then use a Chernoff bound argument. We now present the formal proof.

Proof of Lemma 2.12. Intuitively, this argument is simple, since the connectors are spread out in steps of length approximately λ. However, there might be some *periodicity* that results in the same node being visited multiple times but *exactly* at λ-intervals. This is where we crucially use the fact that the algorithm uses walks of length $\lambda + r$ where r is chosen uniformly at random from $[0, \lambda - 1]$.

We prove the lemma using the following two claims.

Claim 2.13. *Consider any sequence A of numbers $a_1, ..., a'_\ell$ of length ℓ'. For any integer λ', let B be a sequence $a_{\lambda'+r_1}, a_{2\lambda'+r_1+r_2}, ..., a_{i\lambda'+r_1+...+r_i}, ...$ where r_i, for any i, is a random integer picked uniformly from $[0, \lambda' - 1]$. Consider another subsequence of numbers C of A where an element in C is picked from from "every λ' numbers" in A; i.e., C consists of $\lfloor \ell'/\lambda' \rfloor$ numbers $c_1, c_2, ...$ where, for any i, c_i is chosen uniformly at random from $a_{(i-1)\lambda'+1}, a_{(i-1)\lambda'+2}, ..., a_{i\lambda'}$. Then, $Pr[C$ contains $a_{i_1}, a_{i_2}, ..., a_{i_k}\}] = Pr[B = \{a_{i_1}, a_{i_2}, ..., a_{i_k}\}]$ for any set $\{a_{i_1}, a_{i_2}, ..., a_{i_k}\}$.*

Proof. First consider a subsequence C of A. Numbers in C are picked from "every λ' numbers" in A; i.e., C consists of $\lfloor \ell'/\lambda' \rfloor$ numbers $c_1, c_2, ...$ where, for any i, c_i is chosen uniformly at random from $a_{(i-1)\lambda'+1}, a_{(i-1)\lambda'+2}, ..., a_{i\lambda'}$. Observe that $|C| \geq |B|$. In fact, we can say that "C contains B"; i.e., for any sequence of k indexes $i_1, i_2, ..., i_k$ such that $\lambda' \leq i_{j+1} - i_j \leq 2\lambda' - 1$ for all j,

$$Pr[B = \{a_{i_1}, a_{i_2}, ..., a_{i_k}\}] = Pr[C \text{ contains } \{a_{i_1}, a_{i_2}, ..., a_{i_k}\}].$$

To see this, observe that B will be equal to $\{a_{i_1}, a_{i_2}, ..., a_{i_k}\}$ only for a specific value of $r_1, r_2, ..., r_k$. Since each of $r_1, r_2, ..., r_k$ is chosen uniformly at random from $[1, \lambda']$, $Pr[B = \{a_{i_1}, a_{i_2}, ..., a_{i_k}\}] = \lambda'^{-k}$. Moreover, the C will contain $a_{i_1}, ai_2, ..., a_{i_k}\}$ if and only if, for each j, we pick a_{i_j} from the interval that contains it (i.e., from $a_{(i'-1)\lambda'+1}, a_{(i'-1)\lambda'+2}, ..., a_{i'\lambda'}$, for some i'). (Note that $a_{i_1}, a_{i_2}, ...$ are all in different intervals because $i_{j+1} - i_j \geq \lambda'$ for all j.) Therefore, $Pr[C$ contains $a_{i_1}, a_{i_2}, ..., a_{i_k}\}] = \lambda'^{-k}$. □

Claim 2.14. *Consider any sequence A of numbers $a_1, ..., a'_\ell$ of length ℓ'. Consider subsequence of numbers C of A where an element in C is picked from from "every λ' numbers" in A; i.e., C consists of $\lfloor \ell'/\lambda' \rfloor$ numbers $c_1, c_2, ...$ where, for any i, c_i is chosen uniformly at random from $a_{(i-1)\lambda'+1}, a_{(i-1)\lambda'+2}, ..., a_{i\lambda'}$.. For any number x, let n_x be the number of appearances of x in A; i.e., $n_x = |\{i \mid a_i = x\}|$. Then, for any $R \geq 6n_x/\lambda'$, x appears in C more than R times with probability at most 2^{-R}.*

Proof. For $i = 1, 2, ..., \lfloor \ell'/\lambda' \rfloor$, let X_i be a 0/1 random variable that is 1 if and only if $c_i = x$ and $X = \sum_{i=1}^{\lfloor \ell'/\lambda' \rfloor} X_i$. That is, X is the number of appearances of x in C. Clearly, $E[X] = n_x/\lambda'$. Since X_i's are independent, we can apply the Chernoff bound (e.g., in [18, Theorem 4.4.]): For any $R \geq 6E[X] = 6n_x/\lambda'$,

$$Pr[X \leq R] \geq 2^{-R}.$$

The claim is thus proved. □

Now we use the claim to prove the lemma. Choose $\ell' = \ell$ and $\lambda' = \lambda$ and consider any node v that appears at most t times. The number of times it appears as a connector node is the number of times it appears in the subsequence B described in the claim. By applying the claim with $R = t(\log n)^2$, we have that v appears in B more than $t(\log n)^2$ times with probability at most $1/n^2$ as desired. □

Now we are ready to prove Theorem 2.7.

Proof of Theorem 2.7. First, we claim, using Lemma 2.8 and 2.12, that each node is used as a connector node at most $\frac{24d(x)\sqrt{\ell}(\log n)^3}{\lambda}$ times with probability at least $1-2/n$. To see this, observe that the claim holds if each node x is visited at most $t(x) = 24d(x)\sqrt{\ell+1}\log n$ times and consequently appears as a connector node at most $t(x)(\log n)^2/\lambda$ times. By Lemma 2.8, the first condition holds with probability at least $1-1/n$. By Lemma 2.12 and the union bound over all nodes, the second condition holds with probability at least $1-1/n$, provided that the first condition holds. Therefore, both conditions hold together with probability at least $1-2/n$ as claimed.

Now, we choose $\eta = 1$ and $\lambda = 24\sqrt{\ell D}(\log n)^3$. By Lemma 2.1, Phase 1 finishes in $\tilde{O}(\lambda\eta) = \tilde{O}(\sqrt{\ell D})$ rounds with high probability. For Phase 2, SAMPLE-DESTINATION is invoked $O(\frac{\ell}{\lambda})$ times (only when we stitch the walks) and therefore, by Lemma 2.4, contributes $O(\frac{\ell D}{\lambda}) = \tilde{O}(\sqrt{\ell D})$ rounds. Finally, we claim that GET-MORE-WALKS is never invoked, with probability at least $1-2/n$. To see this, recall our claim above that each node is used as a connector node at most $\frac{24d(x)\sqrt{\ell}(\log n)^3}{\lambda}$ times. Moreover, observe that we have prepared this many walks in Phase 1; i.e., after Phase 1, each node has $\eta\lambda d(x) = \frac{24d(x)\sqrt{\ell}(\log n)^3}{\lambda}$ short walks. The claim follows.

Therefore, with probability at least $1-2/n$, the rounds are $\tilde{O}(\sqrt{\ell D})$ as claimed. □

2.4 Extension to Computing k Random Walks

We now consider the scenario when we want to compute k walks of length ℓ from different (not necessary distinct) sources $s_1, s_2, \ldots, s_k$. We show that SINGLE-RANDOM-WALK can be extended to solve this problem. Consider the following algorithm.

MANY-RANDOM-WALKS**:** Let $\lambda = (24\sqrt{k\ell D+1}\log n + k)(\log n)^2$ and $\eta = 1$. If $\lambda > \ell$ then run the naive random walk algorithm, i.e., the sources find walks of length ℓ simultaneously by sending tokens. Otherwise, do the following. First, modify Phase 2 of SINGLE-RANDOM-WALK to create multiple walks, one at a time; i.e., in the second phase, we stitch the short walks together to get a walk of length ℓ starting at s_1 then do the same thing for s_2, s_3, and so on. We state the theorem below.

Theorem 2.15. MANY-RANDOM-WALKS *finishes in* $\tilde{O}\left(\min(\sqrt{k\ell D}+k, \sqrt{k\ell}+k+\ell)\right)$ *rounds with high probability.*

Proof. First, consider the case where $\lambda > \ell$. In this case, $\min(\sqrt{k\ell D}+k, \sqrt{k\ell}+k+\ell) = \tilde{O}(\sqrt{k\ell}+k+\ell)$. By Lemma 2.8, each node x will be visited at most $\tilde{O}(d(x)(\sqrt{k\ell}+k))$ times. Therefore, using the same argument as Lemma 2.1, the congestion is $\tilde{O}(\sqrt{k\ell}+k)$ with high probability. Since the dilation is ℓ, MANY-RANDOM-WALKS takes $\tilde{O}(\sqrt{k\ell}+k+\ell)$ rounds as claimed.

Now, consider the other case where $\lambda \leq \ell$. In this case, $\min(\sqrt{k\ell D}+k, \sqrt{k\ell}+k+\ell) = \tilde{O}(\sqrt{k\ell D}+k)$. Phase 1 takes $\tilde{O}(\lambda\eta) = \tilde{O}(\sqrt{k\ell D}+k)$. The stitching in Phase 2 takes $\tilde{O}(k\ell D/\lambda) = \tilde{O}(\sqrt{k\ell D})$. Moreover, by Lemma 2.8, GET-MORE-WALKS will never be invoked. Therefore, the total number of rounds is $\tilde{O}(\sqrt{k\ell D}+k)$ as claimed. □

3 Lower bound

In this section, we show an almost tight lower bound on the time complexity of performing a distributed random walk. We show that any distributed algorithm needs at least $\Omega\left(\sqrt{\frac{\ell}{\log \ell}}\right)$ rounds, even in graphs with low diameter. Note that $\Omega(D)$ is a lower bound [6]. Also note that if a source node wants to sample k destinations from independent random walks, then $\Omega(k)$ is also a lower bound as the source may need to receive $\Omega(k)$ distinct messages. Therefore, for k walks, the lower bound we show is $\Omega(\sqrt{\frac{\ell}{\log \ell}}+k+D)$ rounds. (The rest of the section omits the $\Omega(k+D)$ term.) In particular, we show that there exists a n-node

graph of diameter $O(\log n)$ such that any distributed algorithm needs at least $\Omega(\sqrt{\frac{n}{\log n}})$ time to perform a walk of length n. Our lower bound proof makes use of a lower bound for another problem that we call as the *Path Verification problem* defined as follows. Informally, the Path Verification problem is for some node v to verify that a given sequence of nodes in the graph is a valid path of length ℓ.

Definition 3.1 (PATH-VERIFICATION Problem)**.** The input of the problem consists of an integer ℓ, a graph $G = (V, E)$, and ℓ nodes $v_1, v_2, ..., v_\ell$ in G. To be precise, each node v_i initially has its order number i.

The goal is for some node v to "verify" that the above sequence of vertices forms an ℓ-length path, i.e., if (v_i, v_{i+1}) forms an edge for all $1 \leq i \leq \ell - 1$. Specifically, v should output "yes" if the sequence forms an ℓ-length path and "no" otherwise.

We show a lower bound for the Path Verification problem that applies to a very general class of verification algorithms defined as follows. Each node can (only) send a segment of the path that it can verify either directly or indirectly (by learning from its neighbors), as follows. Each node can (only) verify a segment of the path that it knows either directly or indirectly (by learning form its neighbors), as follows. Initially each node knows only the trivial segment itself. If a vertex obtains from its neighbor a segment $[i_1, j_1]$ and it has already verified segment $[i_2, j_2]$ that overlaps with $[i_1, j_1]$ (say, $i_1 < i_2 < j_1 < j_2$) then it can verify a larger interval ($[i_1, j_2]$). Note that a node needs to only send the endpoints of the interval that it already verifies (hence larger intervals are better). (See Figure 1 in the Appendix for an example.) The goal of the problem is that, in the end, some node verifies the entire segment $[1, \ell]$. We would like to determine a lower bound for the running time of any distributed algorithm for the above problem.

A lower bound for the Path Verification problem, implies a lower bound for the random walk problem as well. The reason is as follows. Both problems involve constructing a path of some specified length ℓ. Intuitively, the former is a simpler problem, since we are not verifying whether the local steps are chosen randomly, but just whether the path is valid and is of length ℓ. On the other hand, any algorithm for the random walk problem (including our algorithm of Section 2), also solves the Path Verification problem, since the path it constructs should be a valid path of length ℓ. It is straightforward to make any distributed algorithm that computes a random walk to also verify that indeed the random walk is a valid walk of appropriate length. This is essential for correctness, as otherwise, an adversary can always change simply one edge of the graph and ensure that the walk is wrong.

In the next section we first prove a lower bound for the Path Verification problem. The we show the same lower bound holds for the random walk problem by giving a reduction.

3.1 Lower Bound for the Path Verification Problem

The main result of this section is the following theorem.

Theorem 3.2. *For every n, and $\ell \leq n$ there exists a graph G_n of $\Theta(n)$ vertices and diameter $O(\log n)$, and a path P of length ℓ such that any algorithm that solves the* PATH-VERIFICATION *problem on G_n and P requires more than k rounds, where $k = \sqrt{\frac{\ell}{\log \ell}}$.*

The rest of the section is devoted to proving the above Theorem. We start by defining G_n.

Definition 3.3 (Graph G_n)**.** Let k' be an integer such that k is a power of 2 and $k'/2 \leq 4k < k'$. Let n' be such that $n' \geq n$ and k' divides n'. We construct G_n having $(n' + 2k' - 1) = O(n)$ nodes as follows. First, we construct a path $P = v_1 v_2 ... v_{n'}$. Second, we construct a binary T having k' leaf nodes. Let $u_1, u_2, ..., u_{k'}$ be its leaves from left to right. Finally, we connect P with T by adding an edge $u_i v_{jk'+i}$ for every i and j. We will denote the root of T by x and its left and right children by l and r respectively. Clearly, G_n has diameter $O(\log n)$. We then consider a path of length $\ell = \Theta(n)$. If required n can always be made larger by connecting dummy vertices to the root of T. (The resulting graph G_n is as in Figure 3 in the Appendix.) □

To prove the theorem, let $\mathcal{A}$ be any algorithm for the PATH-VERIFICATION problem that solves the problem on G_n in at most k' rounds. We need some definitions and claims to prove the theorem.

Definitions of *left/right subtrees* and *breakpoints*. Consider a tree T' obtained by deleting all edges in P. Notice that nodes $v_{jk'+i}$, for all j and $i \leq k'/2$ are in the subtree of T' rooted at l and all remaining points are in the subtree rooted at r. For any node v, let $sub(v)$ denote the subtree rooted at node v. (Note that $sub(v)$ also include nodes in the path P.) We denote the set of nodes that are leaves of $sub(l)$ by L (i.e., $L = sub(l) \cap P$) and the set of nodes that are leaves in $sub(r)$ by R.

Since we consider an algorithm that takes at most k rounds, consider the situation when the algorithm is given k rounds for *free* to communicate only along the edges of the path P at the beginning. Since L and R consists of every $k'/2$ vertices in P and $k'/2 > 2k$, there are some nodes unreachable from L by walking on P for k steps. In particular, all nodes of the form $v_{jk'+k'/2+k+1}$, for all j, are not reachable from L. We call such nodes *breakpoints* for $sub(l)$. Similarly all nodes of the form $v_{jk'+k+1}$, for all j, are not reachable from R and we call them the breakpoints for $sub(r)$. (See Figure 4 in the Appendix.)

Definitions of *path-distance* and *covering*. For any two nodes u and v in T' (obtained from G_n by deleting edges in P), let $c(u, v)$ be a lowest common ancestor of u and v. We define $path\_dist(u, v)$ to be the number of leaves of subtree of T rooted at $c(u, v)$. Note that the path-distance is defined between any pair of nodes in G_n but the distance is counted using the number of leaves in T (which excludes nodes in P). (See Figure 5(a) in Appendix.)

We also introduce the notion of the path-distance *covered* by a message. For any message m, the path-distance covered by m is the maximum path-distance taken over all nodes that have held the message m. That is, if m covers some nodes $v'_1, v'_2, ..., v'_k$ then the path-distance covered by m is the number of leaves in the subtrees of T rooted by $v'_1, v'_2, ..., v'_k$. Note that some leaves may be in more than one subtrees and they will be counted only once. Our construction makes the right and left subtrees have a large number of break points, as in the following lemma.

Lemma 3.4. *The number of breakpoints for the left subtree and for the right subtree are at least $\frac{n}{4k}$ each.*

Proof. After the first k free rounds, consider the intervals that the left subtree can have, in the best case. Recall that these k rounds allowed communication only along the path. The $path\_dist$ of any node in L from the breakpoints of $sub(L)$ along the path is at least $k + 1$. □

The reason we define these breakpoints is to show that the entire information held by the left subtree has many disjoint intervals, and same for the right subtree. This then tells us that the left subtree and the right subtree must *communicate* a lot to be able to merge these intervals by connecting/communicating the break points. To argue this, we show that the total path distance (over all messages) is large, as in the following lemma.

Lemma 3.5. *For algorithm $\mathcal{A}$ to solve* PATH-VERIFICATION *problem, the total path-distance covered by all messages is at least n.*

Proof. First, notice that each left breakpoint is at a path-distance of $k+1$ from every node in the right subtree. That is, $path\_dist(u, L) = path\_dist(v, R) = k + 1$ for all $u \in B_l$ and all $v \in B_r$.

Each breakpoint needs to be combined into one interval in the end. However, there could be one interval that is communicated from the $sub(l)$ to the $sub(r)$ (or vice versa) such that it connects several breakpoints. We show that this cannot happen. Consider all the breakpoints $v \in B_l \cup B_r$.

Definition of *scratching*.

Let us say that we *scratch out* the breakpoints from the list $k+1$, $k'/2+k+1$, $k'+k+1$, $k'+k'/2+k+1$, $2k' + k + 1$, ... that get connected when an interval is communicated between $sub(l)$ and $sub(r)$. We scratch out a breakpoint if there is an interval in the graph that contains it and both (or one in case of the first and last breakpoints) its adjacent breakpoints. For example, if the left subtree has intervals $[1, k'/2 + k]$ and $[k'/2+k+2, k'+k'/2+k+1]$ and the right subtree has $[k+2, k'+k]$ and the latter interval is communicated to a node in the left subtree, then the left subtree is able to obtain the merged interval $[1, k' + k'/2 + k + 1]$ and therefore breakpoints $k + 1$ and $k'/2 + k + 1$ are scratched out.

Claim 3.6. *At most $O(1)$ breakpoints can be scratched out with one message/interval communicated between $sub(r)$ and $sub(l)$*

Proof. We argue that with the communication of one interval across the left and right subtrees, at most 4 breakpoints that have not been scratched yet can get scratched. This follows from a simple inductive argument. Consider a situation where the left subtree has certain intervals with all overlapping intervals already merged, and similarly right subtree. Suppose an interval $\mathcal{I}$ is communicated between $sub(r)$ and $sub(l)$, one of the following cases arise:

- $\mathcal{I}$ contains one breakpoint: Can be merged with at most two other intervals. Therefore, at most three breakpoints can get scratched.
- $\mathcal{I}$ contains two breakpoints: Can get connected with at most two other intervals and therefore at most four breakpoints can get scratched.
- $\mathcal{I}$ contains more than two breakpoints: This is impossible since there are at most two breakpoints in each interval, its left most and right most numbers (by definition of scratching).

This completes the proof of the claim. □

The proof now follows from Lemma 3.4. For any breakpoint b, let M_b be the set of messages that represents an interval containing b while b is still unscratched. If b is in $sub(l)$ and gets scratched because of the combination of some intervals in $sub(r)$, then we claim that M_b has covered a path-distance of at least k. (Define the path-distance covered by M_b by the total path-distance covered by all messages in M_b.) This is because $b = v_i$ (say), being a breakpoint in $sub(l)$ has i equal to $(k + 1 \mod k')$. Therefore, b is at a path distance of at least k from any node in R. Consequently, b is at a path-distance of at least k from any node in $sub(r)$. Since there are $\Theta(\frac{n}{4k})$ breakpoints, and for any interval to be communicated across the left and right subtree, a path-distance of k must be covered, in total, $\Theta(n)$ path-distance must be covered for all breakpoints to be scratched. This follows from three main observations:

- As shown above, for any breakpoint to be scratched, an interval with a breakpoint must be communicated from $sub(l)$ to $sub(r)$ or vice versa (thereby all messages m containing the breakpoint together covering a path-distance of at least k)
- Any message/interval with unscratched breakpoints has at most two unscratched breakpoints
- As shown in Claim 3.6, at most four breakpoints can be scratched when two intervals are merged.

  The proof follows. (Also see Figure 5(b) for the idea of this proof.) □

These messages can however be communicated using the tree edges as well. We bound the maximum communication that can be achieved across $sub(l)$ and $sub(r)$ indirectly by bounding the maximum path-distance that can be covered in each round. In particular, we show the following lemma.

Lemma 3.7. *In k rounds, all messages together can cover at most a path-distance of $O(k^2 \log k)$.*

Proof. We consider the total number of messages that can go through nodes at any level of the graph, starting from level 0 to level $\log k$ under the congest model.

First notice that if a message is passed at level i of the tree, this can cover a $path\_dist$ of at most 2^i. This is because the subtree rooted at a node at level i has 2^i leaves. Further, by our construction, there are $2^{\log(k')-i}$ nodes at level i. Therefore, all nodes at level i together, in a given round of $\mathcal{A}$ can cover a $dist - path$, path distance, of at most $2^i 2^{\log(k')-i} = 4k + 2$. Therefore, over k rounds, the total $path\_dist$ that can be covered in a single level is $k(k')$. Since there are $O(\log k)$ levels, the total $path\_dist$ that can be covered in k rounds over the entire graph is $O(k^2 \log k)$. (See Figure 5(c) in the Appendix.) □

We now describe the proof of the main theorem using these three claims.

Proof of Theorem 3.2. Use Lemmas 3.5 and 3.7 we know that if $\mathcal{A}$ solves PATH-VERIFICATION, then it needs to cover a $path\_dist$ of n, but in k rounds it can only cover a $path\_dist$ of $O(k^2 \log k)$. But this is $o(n)$ since $k = \sqrt{\frac{n}{\log n}}$, contradiction. □

3.2 Reduction to Random Walk Problem

We now discuss how the lower bound for the Path Verification problem implies the lower bound of the random walk problem. The main difference between PATH-VERIFICATION problem and the random walk problem is that in the former we can specify which path to verify while the latter problem generates different path each time. We show that the "bad" instance (G_n and P) in the previous section can be modified so that with high probability, the generated random walk is "hard" to verify. The theorems below are stated for ℓ length walk/path instead of n as above. As previously stated, if it is desired that ℓ be $o(n)$, it is always possible to add dummy nodes.

Theorem 3.8. *For any n, there exists a graph G_n of $\Theta(n)$ vertices and diameter $O(\log n)$, and $\ell = \Theta(n)$ such that, with high probability, a random walk of length ℓ needs $\Omega(\sqrt{\frac{\ell}{\log \ell}})$ rounds.*

Proof. First, note that Theorem 3.2 can be generalized to the case where the path P has infinite capacity, as follows.

Theorem 3.9. *For any n and $\ell = \Theta(n)$, there exists a graph G_n of $O(n)$ vertices and diameter $O(\log n)$, and a path P of length ℓ such that any algorithm that solves the* PATH-VERIFICATION *problem on G_n and P requires more than $\Omega(sqrt\frac{\ell}{\log \ell})$ rounds, even if edges in P have large capacity (i.e., one can send larger sized messages in one step).*

Proof. This is because the proof of Theorem 3.2 only uses the congestion of edges in the tree T (imposed above P) to argue about the number of rounds. □

Now, we modify G_n to G'_n as follows. Recall that the path P in G_n has vertices $v_1, v_2, ..., v_{n'}$. For each $i = 1, 2, ..., n'$, we define the weight of an edge (v_i, v_{i+1}) to be $(2n)^{2i}$ (note that weighted graphs are equivalent to unweighted multigraphs in our model). By having more weight, these edges have more capacity as well. However, increasing capacity does not affect the claim as shown above. Observe that, when the walk is at the node v_i, the probability of walk will take the edge (v_i, v_{i+1}) is at least $1 - \frac{1}{n^2}$. Therefore, P is the resulting random walk with probability at least $1 - 1/n$. When the random walk path is P, it takes at least $\sqrt{\frac{n}{\log n}}$ rounds to verify, by Theorem 3.9. This completes the proof. We remark that this construction requires exponential in n number of edges (multiedges). For the distributed computing model, this only translates to a larger bandwidth. The length ℓ is still comparable to the number of nodes. □

4 Estimating Mixing Time

We now present an algorithm to estimate the mixing time of a graph from a specified source. Throughout this section, we assume that the graph is connected and non-bipartite (the conditions under which mixing time is well-defined). The main idea in estimating the mixing time is, given a source node, to run many random walks of length ℓ using the approach described in the previous section, and use these to estimate the distribution induced by the ℓ-length random walk. We then compare the distribution at length ℓ, with the stationary distribution to determine if they are *close*, and if not, double ℓ and retry. For this approach, one issue that we need to address is how to compare two distributions with few samples efficiently (a well-studied problem). We introduce some definitions before formalizing our approach and theorem.

Definition 4.1 (Distribution vector)**.** Let $\pi_x(t)$ define the probability distribution vector reached after t steps when the initial distribution starts with probability 1 at node x. Let π denote the stationary distribution vector.

Definition 4.2 ($\tau^x(\epsilon)$ and τ^x_{mix}, mixing time for source x)**.** Define $\tau^x(\epsilon) = \min t : ||\pi_x(t) - \pi||_1 < \epsilon$. Define $\tau^x_{mix} = \tau^x(1/2e)$.

The goal is to estimate τ^x_{mix}. Notice that the definition of τ^x_{mix} is consistent due to the following standard monotonicity property of distributions.

Lemma 4.3. $||\pi_x(t+1) - \pi||_1 \leq ||\pi_x(t) - \pi||_1$.

Proof. The monotonicity follows from the fact that $||Ax||_1 \leq ||x||_1$ where A is the transpose of the transition probability matrix of the graph and x is any probability vector. That is, $A(i,j)$ denotes the probability of transitioning from node j to node i. This in turn follows from the fact that the sum of entries of any column of A is 1.

Now let π be the stationary distribution of the transition matrix A. This implies that if ℓ is ϵ-near mixing, then $||A^l u - \pi||_1 \leq \epsilon$, by definition of ϵ-near mixing time. Now consider $||A^{l+1}u - \pi||_1$. This is equal to $||A^{l+1}u - A\pi||_1$ since $A\pi = \pi$. However, this reduces to $||A(A^l u - \pi)||_1 \leq \epsilon$. It follows that $(\ell + 1)$ is ϵ-near mixing. □

To compare two distributions, we use the technique of Batu et. al. [2] to determine if the distributions are ϵ-near. Their result (slightly restated) is summarized in the following theorem.

Theorem 4.4 ([2])**.** *For any ϵ, given $\tilde{O}(n^{1/2}poly(\epsilon^{-1}))$ samples of a distribution X over $[n]$, and a specified distribution Y, there is a test that outputs PASS with high probability if $|X - Y|_1 \leq \frac{\epsilon^3}{4\sqrt{n}\log n}$, and outputs FAIL with high probability if $|X - Y|_1 \geq 6\epsilon$.*

We now give a very brief description of the algorithm of Batu et. al. [2] to illustrate that it can in fact be simulated on the distributed network efficiently. The algorithm partitions the set of nodes in to buckets based on the steady state probabilities. Each of the $\tilde{O}(n^{1/2}poly(\epsilon^{-1}))$ samples from X now falls in one of these buckets. Further, the actual count of number of nodes in these buckets for distribution Y are counted. The exact count for Y for at most $\tilde{O}(n^{1/2}poly(\epsilon^{-1}))$ buckets (corresponding to the samples) is compared with the number of samples from X; these are compared to determine if X and Y are close. We refer the reader to their paper [2] for a precise description.

Our algorithm starts with $\ell = 1$ and runs $K = \tilde{O}(\sqrt{n})$ walks of length ℓ from the specified source x. As the test of comparison with the steady state distribution outputs FAIL (for choice of $\epsilon = 1/12e$), ℓ is doubled. This process is repeated to identify the largest ℓ such that the test outputs FAIL with high probability and the smallest ℓ such that the test outputs PASS with high probability. These give lower and upper bounds on the required τ^x_{mix} respectively. Our resulting theorem is presented below and the proof follows.

Theorem 4.5. *Given a graph with diameter D, a node x can find, in $\tilde{O}(n^{1/2} + n^{1/4}\sqrt{D\tau^x(\epsilon)})$ rounds, a time $\tilde{\tau}^x_{mix}$ such that $\tau^x_{mix} \leq \tilde{\tau}^x_{mix} \leq \tau^x(\epsilon)$, where $\epsilon = \frac{1}{6912e\sqrt{n}\log n}$.*

Proof. For undirected unweighted graphs, the stationary distribution of the random walk is known and is $\frac{deg(i)}{2m}$ for node i with degree $deg(i)$, where m is the number of edges in the graph. If a source node in the network knows the degree distribution, we only need $\tilde{O}(n^{1/2}poly(\epsilon^{-1}))$ samples from a distribution to compare it to the stationary distribution. This can be achieved by running MULTIPLERANDOMWALK to obtain $K = \tilde{O}(n^{1/2}poly(\epsilon^{-1}))$ random walks. We choose $\epsilon = 1/12e$. To find the approximate mixing time, we try out increasing values of l that are powers of 2. Once we find the right consecutive powers of 2, the monotonicity property admits a binary search to determine the exact value for the specified ϵ.

The result in [2] can also be adapted to compare with the steady state distribution even if the source does not know the entire distribution. As described previously, the source only needs to know the *count* of number of nodes with steady state distribution in given buckets. Specifically, the buckets of interest are at most $\tilde{O}(n^{1/2}poly(\epsilon^{-1}))$ as the count is required only for buckets were a sample is drawn from. Since each node

knows its own steady state probability (determined just by its degree), the source can broadcast a specific bucket information and recover, in $O(D)$ steps, the count of number of nodes that fall into this bucket. Using the standard upcast technique previously described, the source can obtain the bucket count for each of these at most $\tilde{O}(n^{1/2}poly(\epsilon^{-1}))$ buckets in $\tilde{O}(n^{1/2}poly(\epsilon^{-1}) + D)$ rounds.

We have shown previously that a source node can obtain K samples from K independent random walks of length ℓ in $\tilde{O}(K + \sqrt{Kl}D)$ rounds. Setting $K = \tilde{O}(n^{1/2}poly(\epsilon^{-1}) + D)$ completes the proof. □

Suppose our estimate of τ^x_{mix} is close to the mixing time of the graph defined as $\tau_{mix} = \max_x \tau^x_{mix}$, then this would allow us to estimate several related quantities. Given a mixing time τ_{mix}, we can approximate the spectral gap $(1 - \lambda_2)$ and the conductance (Φ) due to the known relations that $\frac{1}{1-\lambda_2} \leq \tau_{mix} \leq \frac{\log n}{1-\lambda_2}$ and $\Theta(1 - \lambda_2) \leq \Phi \leq \Theta(\sqrt{1 - \lambda_2})$ as shown in [11].

5 Concluding Remarks

This paper essentially resolves the time complexity of distributed computation of random walks in undirected networks by showing almost tight upper and lower bounds. The dependence on the diameter D is still not tight, and it would be interesting to settle this. There is also a gap in our bounds for performing k independent random walks. Further, we look at the congest model enforcing a bandwidth restriction and minimize number of rounds. The total message complexity of our algorithms are high. It would be nice to come up with algorithms that are round efficient and yet have smaller message complexity. Another interesting open question is whether the source x based mixing time of the graph can be estimated more efficiently, in particular in $\tilde{O}(\sqrt{\tau^x_{mix}} + n^{1/4})$ rounds? Currently our techniques require roughly $n^{1/4}$ factor more rounds. Further, can our techniques be extended to obtain fast decentralized algorithms to compute the worst case mixing time $\tau_{mix} = \max_x \tau^x_{mix}$ and perhaps the second eigenvector of the transition matrix (which can be used to approximate a sparse cut)?

It will be also interesting to investigate the distributed complexity of performing random walks in directed graphs. This will be useful in the context of developing fast decentralized algorithms for computing PageRank and related quantities. We believe our approach can be useful for doing decentralized computation in large-scale dynamic networks. From a practical standpoint, it is often important to develop algorithms that are robust to failures and therefore it would be nice to extend our techniques to handle such node and edge failures.

References

[1] N. Alon, C. Avin, M. Koucký, G. Kozma, Z. Lotker, and M. R. Tuttle. Many random walks are faster than one. In *SPAA*, pages 119–128, 2008.

[2] T. Batu, E. Fischer, L. Fortnow, R. Kumar, R. Rubenfeld, and P. White. Testing random variables for independence and identity. In *Proc. of the 42nd IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 442–451, 2001.

[3] M. Bui, T. Bernard, D. Sohier, and A. Bui. Random walks in distributed computing: A survey. In *IICS*, pages 1–14, 2004.

[4] C. Cooper, A. Frieze, and T. Radzik. Multiple random walks in random regular graphs. In *Preprint*, 2009.

[5] A. Das Sarma, S. Gollapudi, and R. Panigrahy. Estimating pagerank on graph streams. In *PODS*, pages 69–78, 2008.

[6] A. Das Sarma, D. Nanongkai, and G. Pandurangan. Fast distributed random walks. In *PODC*, 2009.

[7] D. Dubhashi, F. Grandioni, and A. Panconesi. Distributed algorithms via lp duality and randomization. In *Handbook of Approximation Algorithms and Metaheuristics*. 2007.

[8] M. Elkin. An overview of distributed approximation. *ACM SIGACT News Distributed Computing Column*, 35(4):40–57, December 2004.

[9] M. Elkin. Unconditional lower bounds on the time-approximation tradeoffs for the distributed minimum spanning tree problem. In *Proceedings of Symposium on Theory of Computing (STOC)*, June 2004.

[10] C. Gkantsidis, G. Goel, M. Mihail, and A. Saberi. Towards topology aware networks. In *IEEE INFOCOM*, 2007.

[11] M. Jerrum and A. Sinclair. Approximating the permanent. *SIAM Journal of Computing*, 18(6):1149–1178, 1989.

[12] D. Kempe and F. McSherry. A decentralized algorithm for spectral analysis. *Journal of Computer and System Sciences*, 74(1):70–83, 2008.

[13] M. Khan, F. Kuhn, D. Malkhi, G. Pandurangan, and K. Talwar. Efficient distributed approximation algorithms via probabilistic tree embeddings. In *Proc. 27th ACM Symp. on Principles of Distributed Computing (PODC)*, 2008.

[14] M. Khan and G. Pandurangan. A fast distributed approximation algorithm for minimum spanning trees. *Distributed Computing*, 20:391–402, 2008.

[15] S. Kutten and D. Peleg. Fast distributed construction of k-dominating sets and applications. *J. Algorithms*, 28:40–66, 1998.

[16] N. Lynch. *Distributed Algorithms*. Morgan Kaufmann Publishers, San Mateo, CA, 1996.

[17] R. Lyons. Asymptotic enumeration of spanning trees. *Combinatorics, Probability & Computing*, 14(4):491–522, 2005.

[18] M. Mitzenmacher and E. Upfal. *Probability and Computing: Randomized Algorithms and Probabilistic Analysis*. Cambridge University Press, New York, NY, USA, 2005.

[19] G. Pandurangan and M. Khan. Theory of communication networks. In *Algorithms and Theory of Computation Handbook, Second Edition*. CRC Press, 2009.

[20] D. Peleg. *Distributed computing: a locality-sensitive approach*. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 2000.

[21] D. Peleg and V. Rabinovich. A near-tight lower bound on the time complexity of distributed mst construction. In *Proc. of the 40th IEEE Symp. on Foundations of Computer Science*, pages 253–261, 1999.

[22] R. Sami and A. Twigg. Lower bounds for distributed markov chain problems. *CoRR*, abs/0810.5263, 2008.

[23] J. S. Vitter. Random sampling with a reservoir. *ACM Trans. Math. Softw.*, 11(1):37–57, 1985. Also appeared in FOCS'83.

[24] M. Zhong and K. Shen. Random walk based node sampling in self-organizing networks. *Operating Systems Review*, 40(3):49–55, 2006.

Appendix

A Figures

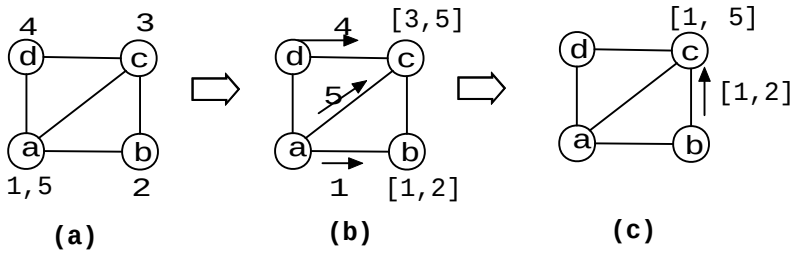


Figure 1: Example of path verification problem. **(a)** In the beginning, we want to verify that the vertices containing numbers $1..5$ form a path. (In this case, they form a path a, b, c, d, a.) **(b)** One way to do this is for a to send 1 to b and therefore b can check that two vertices a and b corresponds to label 1 and 2 form a path. (The interval $[1, 2]$ is used to represent the fact that vertices corresponding to numbers $1, 2$ are verified to form a path.) Similarly, c can verify $[3, 5]$. **(c)** Finally, c combine $[1, 2]$ with $[3, 5]$ and thus the path corresponds to numbers $1, 2, ..., 5$ is verified.

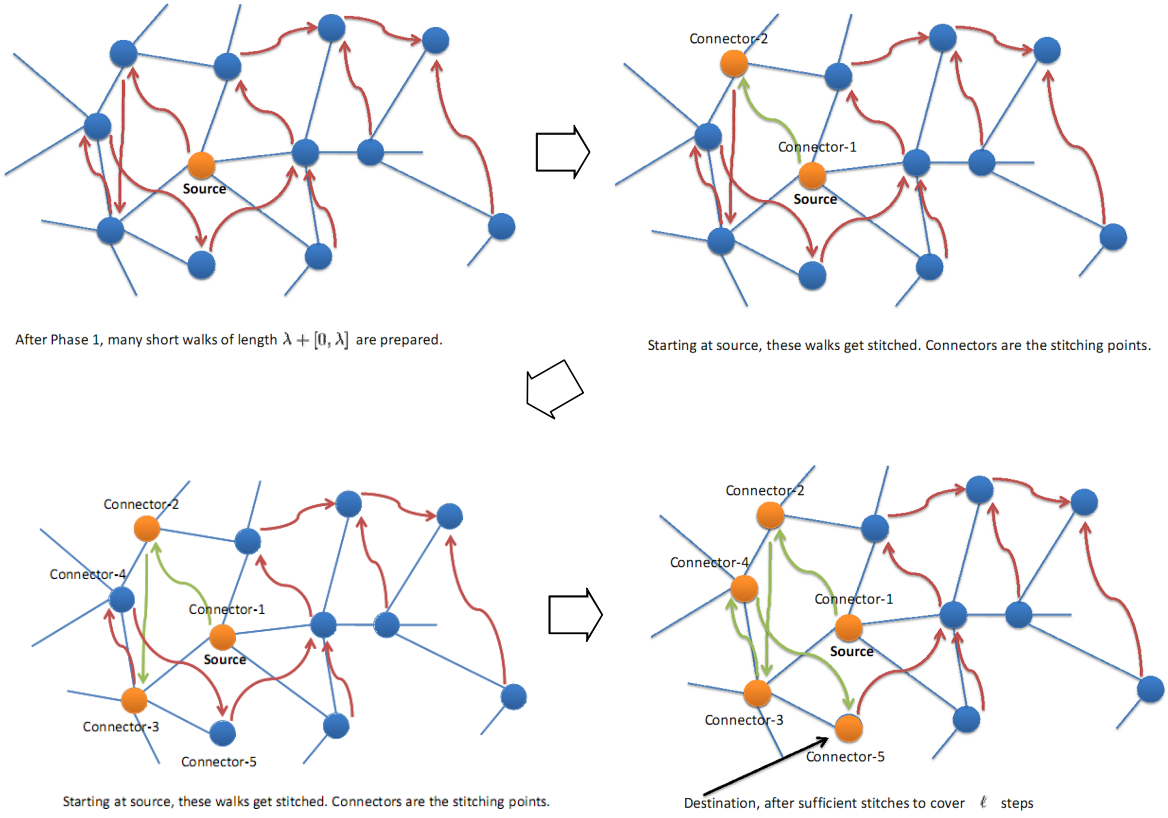


Figure 2: Figure illustrating the Algorithm of stitching short walks together.

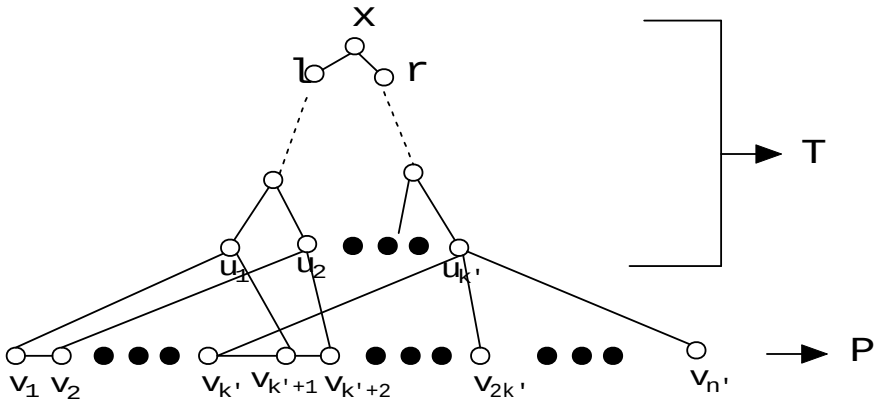
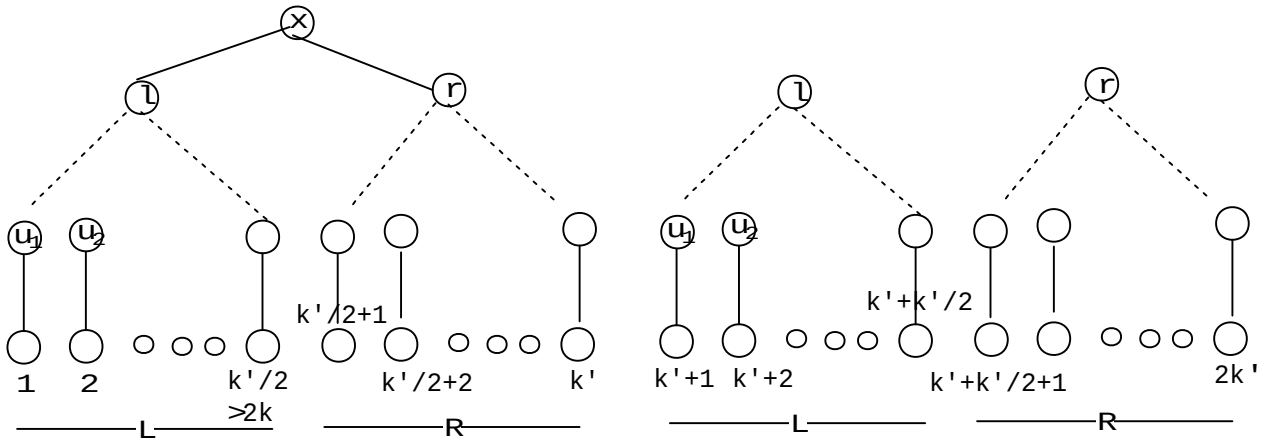


Figure 3: G_n



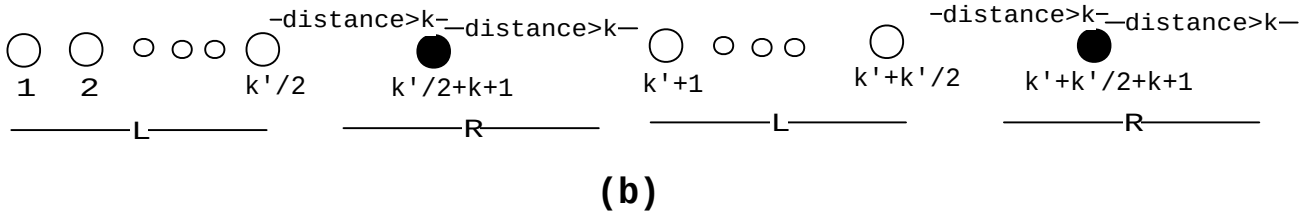
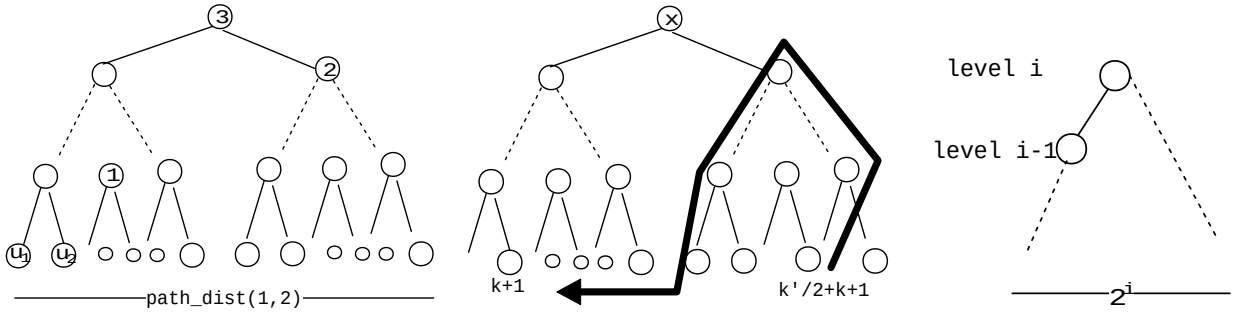


Figure 4: **Breakpoints. (a)** L and R consist of every other $k'/2$ vertices in P. (Note that we show the vertices l and r appear many times for the convenience of presentation.) **(b)** $v_{k'/2+k+1}$ and $v_{k'+k'/2+k+1}$ (nodes in black) are two of the breakpoints for L. Notice that there is one breakpoint in every connected piece of L and R.



(a) Path-distance. (b) Idea of Claim 3.6 (c) Idea of Lemma 3.7.

Figure 5: **(a)** Path distance between 1 and 2 is the number of leaves in the subtree rooted at 3, the lowest common ancestor of 1 and 2. **(b)** For one unscratched left breakpoint, $k'/2+k+1$ to be combined with another right breakpoint $k+1$ on the left, $k'/2+k+1$ has to be carried to L by some intervals. Moreover, one interval can carry at most two unscratched breakpoints at a time. **(c)** Sending a message between nodes on level i and $i-1$ can increase the covered path distance by at most 2^i