

Fast Distributed Random Walks*

Atish Das Sarma
College of Computing
Georgia Institute of
Technology
Atlanta, GA, USA
atish@cc.gatech.edu

Danupon Nanongkai
College of Computing
Georgia Institute of
Technology
Atlanta, GA, USA
danupon@cc.gatech.edu

Gopal Pandurangan
Dept. of Computer Science
Purdue University
West Lafayette, IN 47907,
USA.
gopal@cs.purdue.edu

ABSTRACT

Performing random walks in networks is a fundamental primitive that has found applications in many areas of computer science, including distributed computing. In this paper, 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.

All previous algorithms that compute a random walk sample of length ℓ as a subroutine always do so naively, i.e., in $O(\ell)$ rounds. The main contribution of this paper is a fast distributed algorithm for performing random walks. We show that a random walk sample of length ℓ can be computed in $\tilde{O}(\ell^{2/3}D^{1/3})$ rounds on an undirected unweighted network, where D is the diameter of the network.¹ When $\ell = \Omega(D \log n)$, this is an improvement over the naive $O(\ell)$ bound. (We show that $\Omega(\min\{D, \ell\})$ is a lower bound and hence in general we cannot have a running time faster than the diameter of the graph.) We also show that our algorithm can be applied to speedup the more general Metropolis-Hastings sampling.

We extend our algorithms to perform a large number, k , of random walks efficiently. We show how k destinations can be sampled in $\tilde{O}((k\ell)^{2/3}D^{1/3})$ rounds if $k \leq \ell^2$ and $\tilde{O}((k\ell)^{1/2})$ rounds otherwise. We also present faster algorithms for performing random walks of length larger than (or equal to) the mixing time of the underlying graph. Our techniques can be useful in speeding up distributed algorithms for a variety of applications that use random walks as a subroutine.

Categories and Subject Descriptors

F.2.2 [Analysis of Algorithms, Problem Complexity]: Nonnumerical Algorithms and Problems—*computations on discrete structures*;

*Supported in part by NSF Award CCF-0830476.

¹ \tilde{O} hides $\frac{\log n}{\delta}$ factors where n is the number of nodes in the network and δ is the minimum degree.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

PODC'09, August 10–12, 2009, Calgary, Alberta, Canada.
Copyright 2009 ACM 978-1-60558-396-9/09/08 ...\$10.00.

G.2.2 [Discrete Mathematics]: Graph Theory—*graph algorithms*;

G.2.2 [Discrete Mathematics]: Graph Theory—*network problems*

General Terms

Algorithms, Theory

Keywords

Random walks, Random sampling, Distributed algorithm, Metropolis-Hastings sampling.

1. INTRODUCTION

Random walks play a central role in computer science, spanning a wide range of areas in both theory and practice, including distributed computing. Algorithms in many different applications use random walks as an integral subroutine. Applications in networks include token management [21, 7, 13], load balancing [22], small-world routing [24], search [35, 1, 12, 18, 27], information propagation and gathering [8, 23], network topology construction [18, 25, 26], checking expander [16], constructing random spanning trees [9, 6, 5], monitoring overlays [30], group communication in ad-hoc network [15], gathering and dissemination of information over a network [2], distributed construction of expander networks [25], and peer-to-peer membership management [17, 36]. Random walks have also been used to provide uniform and efficient solutions to distributed control of dynamic networks [10]. The paper of [35] describes a broad range of network applications that can benefit from random walks in dynamic and decentralized settings. For further references on applications of random walks to distributed computing, see, e.g. [10, 35]. A key purpose of random walks in many of these network applications is to perform node sampling. Random walk-based sampling is simple, local, and robust. 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. In this paper we present the first non-trivial distributed random walk sampling algorithms in arbitrary networks that are significantly faster than the existing (naive) approaches.

Problems

Although using random walks help in improving the performance of many distributed algorithms, all known algorithms perform random walks naively: Each walk of length ℓ is performed by sending a token for ℓ steps, picking a random

neighbor with each step. Is there a faster way to perform a random walk distributively? In particular, we consider the following *basic* random walk problem.

Computing One Random Walk where Destination Outputs Source. Let s be any node in the network. We want a distributed algorithm such that, in the end, one node v outputs the ID of s where v is randomly picked according to the probability that it is the destination of a random walk of length ℓ starting at s (the source node). We want an algorithm that finishes in the smallest number of rounds.

We consider the following generalizations to the problem.

1. *k Random Walks, Destinations output Sources (k-RW-DoS):* We have k sources s_1, s_2, \dots, s_k (not necessarily distinct) and we want each of k destinations to output an ID of its corresponding source.
2. *k Random Walks, Sources output Destinations (k-RW-SoD):* Same as above but we want each source to output the ID of its corresponding destination.

It turns out that solving k -RW-SoD can be more expensive than solving k -RW-DoS. An extension of the first problem can be used in applications where the sources only want to know a “synopsis” of the destination, such as aggregating statistics and computing a function (max load, average load) by sampling nodes. The second problem is used when sources want to know data of each destination separately.

To demonstrate that these problems are non-trivial, let us first focus on the basic random walk problem (which is equivalent to 1-RW-DoS). The following naive algorithm finishes in $O(\ell)$ rounds: Circulate a token (with ID of s written on it) starting from s for ℓ rounds (in each round, the node having the token currently, forwards it to a random neighbor) and, in the end, the vertex v that holds the token outputs the ID of s . Our goal is to devise algorithms that are faster than this ℓ -round algorithm. To achieve faster algorithms, a node cannot just wait until it receives the token and forwards it. It is necessary to “forward the token ahead of time”. One natural approach is to guess which nodes will be in the walk and ask them to forward the token ahead of time. However, even if one knew how many times each node is expected to be seen on the walk (without knowing the order), it is still not clear what running time one can guarantee. The difficulty is that many pre-forwarded tokens may cause congestion. A new approach is needed to obtain fast distributed computation of random walks. We present the first such results in this paper.

Notation: Throughout the paper, we let ℓ be the length of the walks, k be the number of walks, D be the network diameter, δ be the minimum node degree and n be the number of nodes in the network.

Distributed Computing Model

Before we present our results, we describe our model which is standard in the literature. Without loss of generality, we assume that the graph is connected. Each node has a unique identifier and at the beginning of the computation, each node v accepts as input its own identifier. The nodes are allowed to communicate (only) through the edges of the graph G . We assume that the communication is synchronous and occurs in discrete rounds (time steps). We assume the *CONGEST* communication model, a widely used standard model to study distributed algorithms [32, 31]: a node v can

send an arbitrary message of size at most $O(\log n)$ through an edge per time step. (We note that if unbounded-size messages were allowed through every edge in each time step, then the problems addressed here can be trivially solved in $O(D)$ time by collecting all the topological information at one node, solving the problem locally, and then broadcasting the results back to all the nodes [32].) It is typically straightforward to generalize the results to a *CONGEST(B)* model, where $O(B)$ bits can be transmitted in a single time step across an edge. Our time bounds (measured in number of rounds) are for the synchronous communication model. However, our algorithms will also work in an asynchronous model under the same asymptotic time bounds, using a standard tool called the *synchronizer* [32]. We assume that all nodes start simultaneously.

Our Main Contributions

We present the first non-trivial distributed algorithms for computing random walks (both k -RW-DoS and k -RW-SoD) in undirected, unweighted graphs. First, for 1-RW-DoS (cf. Section 2), we give an

$O\left(\frac{\ell^{2/3} D^{1/3} (\log n)^{1/3}}{\delta^{1/3}}\right)$ -round algorithm. Many real-world networks (e.g., peer-to-peer networks) have small diameter D and random walks of length at least the diameter are usually performed; that is, $\ell \gg D$. In this case, the algorithm above finishes in roughly $\tilde{O}(\ell^{2/3})$ rounds, which is a significant improvement over the naive $O(\ell)$ round algorithm. The main idea behind our $O\left(\frac{\ell^{2/3} D^{1/3} (\log n)^{1/3}}{\delta^{1/3}}\right)$ -round algorithm is to “prepare” a few short walks in the beginning and carefully stitch these walks together later as necessary. If there are not enough short walks, we construct more of them on the fly. We overcome a key technical problem by showing how one can perform many short walks in parallel without causing too much congestion. Our results also apply to the cost-sensitive model [4] on weighted graphs.

We then present extensions of our algorithm to perform random walk according to the Metropolis-Hastings [20, 28] algorithm, a more general type of random walk with numerous applications (e.g., [35]). The Metropolis-Hastings algorithm gives a way to define transition probabilities so that a random walk converges to any desired distribution. For an important special case, when the desired distribution is uniform, our time bounds reduce to the same as above.

The above algorithms can be extended to solve k -RW-DoS (cf. Section 3) in $O\left(k \cdot \frac{\ell^{2/3} D^{1/3} (\log n)^{1/3}}{\delta^{1/3}}\right)$ rounds straightforwardly. However, we show that, with a small modification, one can do much better. We give algorithms for two cases. When the goal is to perform a few walks, i.e. $k \leq \ell^2$, we can do this in $O\left(\frac{(k\ell)^{2/3} D^{1/3} (\log n)^{1/3}}{\delta^{1/3}}\right)$ rounds. Moreover, when

$k \geq \ell^2$, one can do this in only $O\left(\sqrt{\frac{k\ell \log n}{\delta}}\right)$ rounds. We then give a simple algorithm for an important special case, namely when $\ell \geq t_{mix}$ where t_{mix} is the mixing time of the underlying graph (cf. Section 5). We develop an $O(D + k)$ algorithm for k -RW-DoS and k -RW-SoD. We also observe that $\Omega(\min(\ell, D))$ is a straightforward lower bound. Therefore, we have tight algorithms when $\ell \leq D$ or $\ell \geq t_{mix}$ and, for $D \leq \ell \leq t_{mix}$, we have efficient non-trivial algorithms.

Finally, we extend the k -RW-DoS algorithms to solve the k -RW-SoD problem (cf. Section 4) in additional $O(k)$ rounds. We also observe that $\Omega(k + \min(\ell, D))$ is a lower bound on the number of rounds required. Therefore, our

algorithms for k -RW-SoD are asymptotically tight. We note an interesting fact that algorithms for k -RW-DoS finishes in $o(k)$ rounds when k is much larger than ℓ and D while k is the lower bound of k -RW-SoD.

We note that the focus of this paper is on the time complexity and not on the message (communication) complexity of performing random walks. In general, the message complexity of our algorithms can be larger than the message complexity of the naive random walk algorithm (that takes only ℓ messages to perform a walk of length ℓ).

Applications and Related Work

Random walks have been used in a wide variety of applications in distributed networks as mentioned in the beginning. We describe here some of the applications in more detail.

Speeding up distributed algorithms using random walks has been considered for a long time. Besides our approach of speeding up the random walk itself, one popular approach is to reduce the *cover time*. Recently, Alon et. al. [3] show that performing several random walks in parallel reduces the cover time in various types of graphs. They assert that the problem with performing random walks is often the latency. In these scenarios where many walks are performed, our results could help avoid too much latency and yield an additional speed-up factor.

A nice application of random walks is in the design and analysis of expanders. We mention two results here. Law and Siu [25] consider the problem of constructing expander graphs in a distributed fashion. One of the key subroutines in their algorithm is to perform several random walks from specified source nodes. While the overall running time of their algorithm depends on other factors, the specific step of computing random walk samples can be improved using our techniques presented in this paper. Dolev and Tzachar [16] use random walks to check if a given graph is an expander. The first algorithm given in [16] is essentially to run a random walk of length $n \log n$ and mark every visited vertices. Later, it is checked if every node is visited. It can be seen that our algorithm implies that the first step can be done in $\tilde{O}((n \log n)^{2/3} D^{1/3})$ rounds.

Broder [9] and Wilson [34] gave algorithms to generate random spanning trees using random walks and Broder’s algorithm was later applied to the network setting by Bar-Ilan and Zernik [6]. Recently Goyal et al. [19] show how to construct an expander/sparsifier using random spanning trees. If their algorithm is implemented on a distributed network, the techniques presented in this paper would yield an additional speed-up in the random walk constructions.

Morales and Gupta [30] discuss about discovering a consistent and available monitoring overlay for a distributed system. For each node, one needs to select and discover a list of nodes that would monitor it. The monitoring set of nodes need to satisfy some structural properties such as consistency, verifiability, load balancing, and randomness, among others. This is where random walks come in. Random walks is a natural way to discover a set of random nodes that are spread out (and hence scalable), that can in turn be used to monitor their local neighborhoods. Random walks have been used for this purpose in another paper by Ganesh et al. [17] on peer-to-peer membership management for gossip-based protocols.

The only work that uses the same general approach as this paper is the recent paper of Das Sarma et al. [14]. They con-

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 performs η random walks of length λ)

- 1: Each node constructs η (identical) messages containing its ID and a counter which is initialized to 0.
- 2: **for** $i = 1$ to λ **do**
- 3: 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 $i-1$). Pick 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.}
- 4: **end for**

Phase 2: (Stitch ℓ/λ walks of length λ)

- 1: s creates a message called “token” with the ID of s
 - 2: **for** $i = 1$ to $\lfloor \ell/\lambda \rfloor$ **do**
 - 3: Let v be a node that is currently holding a token
 - 4: v calls `SAMPLE-DESTINATION(v)` and let v' be the returned value (which is a destination of an unused random walk of length λ starting at v)
 - 5: **if** $v' = \text{NULL}$ (all walks from v have already been used up) **then**
 - 6: v calls `GET-MORE-WALKS(v, η, λ)` (Perform η walks of length λ starting at v)
 - 7: v calls `SAMPLE-DESTINATION(v)` and let v' be the returned value
 - 8: **end if**
 - 9: v sends the token to v'
 - 10: **end for**
 - 11: Walk naively until ℓ steps are completed (this is at most another λ steps).
 - 12: A node holding the token outputs the ID of s
-

sider the problem of finding random walks in data streams with the main motivation of finding PageRank. The same general idea of stitching together short walks is used. They consider the model where the graph is too big to store in main memory, and the algorithm has *streaming* access to the edges of the graph while maintaining limited storage. They show how to perform ℓ length random walks in about $\sqrt{\ell}$ passes over the data. This improves upon the naive ℓ pass approach and thereby leads to improved algorithms for estimating PageRank vectors. The distributed setting considered in this paper has very different constraints and motivations from the streaming setting and calls for new techniques. Recently, Sami and Twigg [33] 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.

2. ALGORITHM FOR ONE RANDOM WALK

2.1 Description of the Algorithm

In this section, we present the main ideas of our approach by developing an algorithm for 1-RW-DoS called SINGLE-RANDOM-WALK (cf. Algorithm 1) for undirected graphs.

Algorithm 2 GET-MORE-WALKS(v, η, λ)

(Starting from node v , perform η number of random walks, each of length λ .)

- 1: The node v constructs η (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 , up on receiving ID of v and $c(u, v)$, constructs $c(u, v)$ messages, each contains the ID of v .
 - 7: **end for**
-

The naive upper and lower bounds for 1-RW-DoS are ℓ and D respectively (the lower bound is formalized later in this paper). We present the first nontrivial upper bound, i.e., perform walks of length $\ell > D$ in fewer than ℓ rounds. SINGLE-RANDOM-WALK runs with two important variables: η and λ . The main idea is to first perform η random walks of length λ from every node. Subsequently, starting at the source node s , these λ length walks are “stitched” to form a longer walk (traversing a new walk of length λ from the end point of the previous walk of length λ). Whenever a node is visited as an end point of such a walk of length λ , one of its (at most η) *unused* walks is sampled uniformly to preserve randomness. If all η walks from a node have been used up, additional rounds are invested to obtain η more walks from this node. This approach turns out to be round-efficient for three reasons. First, performing the initial set of η walks of length λ from all nodes simultaneously can be done efficiently. Second, we give a technique to perform η walks of length λ from a single node efficiently. Finally, stitching two λ length walks can be done in about D rounds.

We now explain algorithm SINGLE-RANDOM-WALK (cf. Algorithm 1) in some more detail. The algorithm consists of two phases. In the first phase, each node performs η random walks of length λ each. To do this, each node initially constructs η messages with its ID. Then, each node forwards each message to a random neighbor. This is done for λ steps. At the end of this phase, if node u has k messages with the ID of node v , then u is a destination of k walks starting at v . Note that v has no knowledge of the destinations of its own walks. The main technical issue to deal with here is that performing many simultaneous random walks can cause too much congestion. We show a key lemma (Lemma 2.7) that bounds the time needed for this phase.

In the second phase, we perform a random walk starting from source s by “stitching” walks of length λ obtained in the first phase into a longer walk. The process goes as follows. Imagine that there is a token initially held by s . Among η walks starting at s (obtained in phase 1), randomly select one. Note that this step is not straightforward since s has no knowledge of the destinations of its walks. Further, selecting an arbitrary destination would violate randomness. (A minor technical point: one may try to use the i -th walk when it is reached for the i -th time; however, this is not possible because one cannot mark tokens separately in GET-MORE-

WALKS (described later), since we only send counts forward to avoid congestion on edges). SAMPLE-DESTINATION algorithm (cf. Algorithm 3) is used to perform this step. We prove in Lemma 2.5 that this can be done in $O(D)$ rounds.

When SAMPLE-DESTINATION(v) is called by any node v , the algorithm randomly picks a message v ’s ID written on it, returns the ID of the node that holds this message, and then deletes it. If there is no such message (e.g., when SAMPLE-DESTINATION(v) has been called η times), it returns NULL.

Let v receive u_d as an output from SAMPLE-DESTINATION. Notice that v receives the ID of u_d through the edges on the graph, thereby requiring D rounds. v sends the token to u_d and the process repeats. That is, u_d randomly selects a random walk starting at u_d and forwards the token to the destination. If the process continues without SAMPLE-DESTINATION returning NULL, then a walk of length ℓ will complete after ℓ/λ repetitions.

However, if NULL is returned by SAMPLE-DESTINATION for v , then the token cannot be forwarded further. At this stage, η more walks of length λ are performed from v by calling GET-MORE-WALKS(v, η, λ) (cf. Algorithm 2). This algorithm creates η messages with ID v and forwards them for λ random steps. This is done fast by only sending counts along edges that require multiple messages. This is crucial in avoiding congestion. While one cannot directly bound the number of times any particular node v invokes GET-MORE-WALKS, a simple amortization argument is used to bound the running time of invocations over all nodes.

2.2 Analysis

THEOREM 2.1. *Algorithm SINGLE-RANDOM-WALK (cf. Algorithm 1) solves 1-RW-DoS and, with high probability², finishes in $O(\frac{\ell^{2/3} D^{1/3} (\log n)^{1/3}}{\delta^{1/3}})$ rounds.*

We begin by analyzing the time needed by Phase 1 of Algorithm SINGLE-RANDOM-WALK.

LEMMA 2.2. *Phase 1 finishes in $O(\frac{\lambda \eta \log n}{\delta})$ rounds with high probability, where δ is the minimum node degree.*

PROOF. Consider the case when each node v creates $\eta \cdot \frac{\text{degree}(v)}{\delta} \geq \eta$ messages. We show that the lemma holds even in this case. For each message M , any $j = 1, 2, \dots, \lambda$, 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 2.3. *For any edge e and any j , $\mathbb{E}[X^j(e)] = 2\frac{\eta}{\delta}$.*

PROOF. Assume that each node v starts with $\eta \cdot \frac{\text{degree}(v)}{\delta} \geq \eta$ 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 $\eta \cdot \frac{\text{degree}(v)}{\delta} \geq \eta$. Consider the random walk’s probability transition matrix, call it A . In this case $Au = u$ for the vector u having value $\frac{\text{degree}(v)}{2m}$ 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

²With high probability means with probability at least $(1 - \frac{1}{n})$ throughout this paper.

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 $\text{degree}(x)$. The lemma follows. \square

By Chernoff's bound (e.g., in [29, Theorem 4.4.]), for any edge e and any j ,

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

It follows that the probability that there exists an edge e and an integer $1 \leq j \leq \lambda$ such that $X^j(e) \geq 4 \log n \frac{\eta}{\delta}$ 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 $X^j(e) \leq 4 \log n \frac{\eta}{\delta}$ 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 $4 \log n \frac{\eta}{\delta}$ rounds. Therefore, we obtain walks of length λ in $4\lambda \frac{\eta}{\delta} \log n$ rounds as claimed. (Note that if $\eta \leq \delta$, we still get a high probability bound for $X^j(e) \geq 4 \log n$.) \square

We next show the time needed for GET-MORE-WALKS and SAMPLE-DESTINATION.

LEMMA 2.4. *For any v , GET-MORE-WALKS(v, η, λ) always finishes within $O(\lambda)$ rounds.*

PROOF. 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 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. \square

LEMMA 2.5. *SAMPLE-DESTINATION always finishes within $O(D)$ rounds.*

PROOF. 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. \square

Next we show the correctness of the SAMPLE-DESTINATION algorithm.

LEMMA 2.6. *Algorithm SAMPLE-DESTINATION(v) (cf. Algorithm 3), for any node v , samples a destination of a walk of length λ uniformly at random.*

Algorithm 3 SAMPLE-DESTINATION(v)

Input: Starting node v , and desired walk length λ .

Output: A node sampled from among the stored λ -length walks 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, sampling 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 λ 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 $d_0, d_1, d_2, \dots, d_q$, with counts $c_0, c_1, c_2, \dots, c_q$ (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 $\frac{c_j}{c_0 + c_1 + \dots + c_q}$.
- 7: The sampled token is sent to the parent node (unless already at root), along with a count of $c_0 + c_1 + \dots + c_q$ (the count represents the number of tokens from which this token has been sampled).
- 8: **end for**
- 9: The root outputs 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 (in D rounds through graph edges). u_d deletes one token of v it is holding (so that this random walk of length λ is not reused/re-stitched).
-

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. Suppose that u receives tokens and counts from q children. Assume that it receives tokens d_1, d_2, \dots, d_q and counts c_1, c_2, \dots, c_q from nodes u_1, u_2, \dots, 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 $\frac{1}{|S_{u_j}|} \times \frac{c_j}{c_0+c_1+\dots+c_q} = \frac{1}{S_u}$.

The lemma follows by applying the claim above to v . \square

We are now ready to state and prove the running time of the main algorithm for 1-RW-DoS.

LEMMA 2.7. *Algorithm SINGLE-RANDOM-WALK (cf. Algorithm 1) solves 1-RW-DoS and, with high probability, finishes in $O(\frac{\lambda\eta \log n}{\delta} + \frac{\ell D}{\lambda} + \frac{\ell}{\eta})$ rounds.*

PROOF. First, we prove the correctness of the algorithm. Observe that any two λ -length walks (possibly from different sources) are independent from each other. Moreover, a walk from a particular node is picked uniformly at random (by Lemma 2.6). Therefore, the SINGLE-RANDOM-WALK algorithm is equivalent to having a source node perform a walk of length λ and then have the destination do another walk of length λ and so on.

We now prove the time bound. First, observe that algorithm SAMPLE-DESTINATION is called $O(\frac{\ell}{\lambda})$ times and by Lemma 2.5, this algorithm takes $O(\frac{\ell D}{\lambda})$ rounds in total. Next, we claim that GET-MORE-WALKS is called at most $O(\frac{\ell}{\lambda\eta})$ times in total (summing over all nodes). This is because when a node v calls GET-MORE-WALKS(v, η, λ), all η walks starting at v must have been stitched and therefore v contributes $\lambda\eta$ steps of walk to the long walk we are constructing. It follows from Lemma 2.4 that GET-MORE-WALKS algorithm takes $O(\frac{\ell}{\eta})$ rounds in total.

Combining the above results with Lemma 2.2 gives the claimed bound. \square

Theorem 2.1 immediately follows.

PROOF. Use Lemma 2.7 with $\lambda = \frac{\ell^{1/3} D^{2/3} \delta^{1/3}}{(\log n)^{1/3}}$ and $\eta = \frac{\ell^{1/3} \delta^{1/3}}{D^{1/3} (\log n)^{1/3}}$. \square

2.3 Generalization to the Metropolis-Hastings

We now discuss extensions of our algorithm to perform random walk according to the Metropolis-Hastings algorithm, a more general type of random walk with numerous applications (e.g., [35]). The Metropolis-Hastings [20, 28] algorithm gives a way to define a transition probability so that a random walk converges to any desired distribution π (where π_i , for any node i , is the desired stationary distribution at node i). It is assumed that every node i knows its steady state distribution π_i (and can know its neighbors' steady state distribution in one round). The algorithm is roughly as follows. For any desired distribution π and any desired *laziness factor* $0 < \alpha < 1$, the transition probability from node i to its neighbor j is defined to be $P_{ij} = \alpha \min(1/d_i, \pi_j/(\pi_i d_j))$ where d_i and d_j are degree of i and j respectively. It is shown that a random walk with this transition probability converges to π . We claim that one can modify the SINGLE-RANDOM-WALK algorithm to compute a random walk with above transition probability. We state the main theorem for this process below.

THEOREM 2.8. *The SINGLE-RANDOM-WALK algorithm with transition probability defined by Metropolis-Hastings algorithm finishes in $O(\frac{\lambda\eta \log n}{\min_{i,j}(d_i \pi_j / (\alpha \pi_i))} + \frac{\ell D}{\lambda} + \frac{\ell}{\eta})$ rounds with high probability.*

The proof is similar as in the previous Section. In particular, all lemmas proved earlier hold for this case except Lemma 2.2. We state a similar lemma here with proof

LEMMA 2.9. *For any π and α , Phase 1 finishes in $O(\frac{\lambda\eta \log n}{\min_{i,j}(d_i \pi_j / (\alpha \pi_i))})$ rounds with high probability.*

PROOF. The proof is essentially the same as Lemma 2.2. We present it here for completeness. Let $\beta = \min_i \frac{d_i}{\alpha \pi_i}$ and let $\rho = \frac{\eta}{\beta \min_i \pi_i}$. Consider the case when each node i creates $\rho\beta\pi_i \geq \eta$ messages. We show that the lemma holds even in this case.

We use the same definition as in Lemma 2.2. That is, for each message M , any $j = 1, 2, \dots, \lambda$, 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. As before, we show the following claim.

CLAIM 2.10. *For any edge e and any j , $\mathbb{E}[X^j(e)] = 2\rho$.*

PROOF. Assume that each node v starts with $\rho\beta\pi_i \geq \eta$ 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 $\rho\beta\pi_i$. Consider the random walk's probability transition matrix, say A . In this case $Au = u$ for the vector u having value π_v (since this π_v is the stationary distribution). 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 $\rho\beta\pi_x \times \min(\frac{1}{d_x}, \frac{\pi_y}{\pi_x d_y}) \leq \rho$ (since $\beta \leq \frac{d_x}{\alpha \pi_x}$). The lemma follows. \square

By Chernoff's bound (e.g., in [29, Theorem 4.4.]), for any edge e and any j ,

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

It follows that the probability that there exists an edge e and an integer $1 \leq j \leq \lambda$ such that $X^j(e) \geq 4\rho \log n$ 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 $X^j(e) \leq 4\rho \log n$ 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 $4\rho \log n$ rounds. Therefore, we obtain walks of length λ in $4\lambda\rho \log n = \frac{\lambda\eta \log n}{\min_{i,j}(d_i \pi_j / (\alpha \pi_i))}$ rounds as claimed. \square

An interesting application of the above lemma is when π is a uniform distribution. In this case, we can compute a random walk of length ℓ in $O(\frac{\lambda\eta \alpha \log n}{\delta} + \frac{\ell D}{\lambda} + \frac{\ell}{\eta})$ rounds which is exactly the same as Lemma 2.7 if we use $\alpha = 1$. In both cases, the minimum degree node causes the most congestion. (In each iteration of Phase 1, it sends the same amount of tokens to each neighbor.)

We end this Section by mentioning two further extensions of our algorithm.

1. Weighted graphs: We can generalize our algorithm for weighted graphs, if we assume the cost sensitive communication model of [4]. In this model, we assume that one can

send messages proportional to the weights (so weights serve as bandwidth). This model can be thought of as our model with unweighted multigraph network (note that our algorithm will work for an undirected unweighted multigraph also). The algorithms and analyses are similar.

2. **Obtaining the entire walk:** In the above algorithm, we focus on sampling the destination of the walk and not the more general problem of actually obtaining the entire walk (where every node in the walk knows its position in the walk). However, it is not difficult to extend our approach to the case where each node on the eventual ℓ length walk wants to know its position in the walk within the same time bounds.

3. ALGORITHM FOR K RANDOM WALKS

The previous section is devoted to performing a single random walk of length ℓ efficiently. Many settings usually require a large number of random walk samples. A larger number of samples allows for better estimation of the problem at hand. In this section, we focus on obtaining several random walk samples. For k samples, one could simply run k iterations of the algorithm presented in the previous section; this would require $\tilde{O}(k\ell^{2/3}D^{1/3})$ rounds for k walks. Our algorithms in this section do much better. We consider two different cases, $k \leq \ell^2$ and $k > \ell^2$ and show bounds of $\tilde{O}((k\ell)^{2/3}D^{1/3})$ and $\tilde{O}(\sqrt{k\ell})$ rounds respectively (saving a factor of $k^{1/3}$). Notice, however, that our results still require greater than k rounds. Our algorithms for the two cases are slightly different; the reason we break this in two parts is because in one case we are able to obtain a stronger result than the other. Before that, we first observe a simple lower bound for the k -RW-DoS problem.

LEMMA 3.1. *For every $D \leq n$ and every k , there exists a graph G of diameter D such that any distributed algorithm that solves k -RW-DoS on G uses $\Omega(\min(\ell, D))$ rounds with high probability.*

PROOF. First, consider when $\ell \geq D$. Let s and t be two nodes of distance exactly D from each other and with only this one path between them. A walk of length ℓ starting at s has a non-zero probability of ending up at t . In this case, for the source ID of s to reach t , at least D rounds of communication will be required. Using multi-edges, one can force, with high probability, the traversal of the random walk to be along this D length path. Recall that our algorithms can handle multi-edges. This argument holds for 1-RW-DoS. The constructed multigraph can be changed to a simple graph by subdividing each edge with one additional vertex. This way, in expectation, the walk takes two steps of crossing over from one multiedge to another. Now the same argument can be applied for a random walk of length $O(D)$.

Similarly, if $D \geq \ell$ then we consider s and t of distance exactly ℓ apart and apply the same argument. \square

3.1 k -RW-DoS Algorithm when $k \leq \ell^2$

In this case, the algorithm is essentially repeating algorithm SINGLE-RANDOM-WALK (cf. Algorithm 1) on each source. However, the crucial observation is that we have to do Phase 1 only once.

THEOREM 3.2. *Algorithm FEW-RANDOM-WALKS (cf. Algorithm 4) solves k -RW-DoS for any $k \leq \ell^2$ and, with high probability, finishes in $O(\frac{(k\ell)^{2/3}D^{1/3}(\log n)^{1/3}}{\delta^{1/3}})$ rounds.*

Algorithm 4 FEW-RANDOM-WALKS($\{s_j\}, 1 \leq j \leq k, \ell$)

Input: Starting nodes s_1, s_2, \dots, s_k (not necessarily distinct), and desired walks of length ℓ .

Output: Each destination outputs the ID of its corresponding source.

Phase 1: (Each node performs η random walks of length λ)

- 1: Perform η walks of length λ as in Phase 1 of algorithm SINGLE-RANDOM-WALK (cf. Algorithm 1).

Phase 2: (Stitch ℓ/λ short walks)

- 1: **for** $j = 1$ to k **do**
 - 2: Consider source s_j . Use algorithm SINGLE-RANDOM-WALK to perform a walk of length ℓ from s_j .
 - 3: When algorithm SINGLE-RANDOM-WALK terminates, the sampled destination outputs ID of the source s_j .
 - 4: **end for**
-

PROOF. The first phase of the algorithm finishes in $O(\frac{\lambda\eta \log n}{\delta})$ with high probability using Lemma 2.2 as it is the same as in 1-RW-DoS. The second phase of FEW-RANDOM-WALKS takes rounds $O(k(\frac{\ell D}{\lambda}))$. This follows from the argument in Lemma 2.7. Essentially, the number of times SAMPLE-DESTINATION will be called by k -RW-DoS is at most k times that by 1-RW-DoS; this is $\frac{k\ell}{\lambda}$. Each call requires $O(D)$ rounds. Finally, GET-MORE-WALKS will be called $\frac{k\ell}{\eta\lambda}$ times, again by the argument in Lemma 2.7. Each call requires $O(\lambda)$ rounds, by Lemma 2.4. Combining these, the total number of rounds used is $O(\frac{\lambda\eta \log n}{\delta} + k(\frac{\ell D}{\lambda} + \frac{\ell}{\eta}))$.

To optimize this expression, we choose $\eta = \frac{(k\ell\delta)^{1/3}}{(D \log n)^{1/3}}$ and $\lambda = \frac{(k\ell\delta)^{1/3}D^{2/3}}{(\log n)^{1/3}}$, and the result follows. \square

We now turn to the second case, where the number of walks required, k , exceeds ℓ^2 .

3.2 k -RW-DoS Algorithm when $k \geq \ell^2$

When k is large, our choice of λ becomes ℓ and the algorithm can be simplified to be as in algorithm MANY-RANDOM-WALKS (cf. Algorithm 5). In this algorithm, we do Phase 1 as usual. However, since we use $\lambda = \ell$, we do not have to stitch the short walks together. Instead, we simply check at each source nodes s_i if it has enough walks starting from s_i . If not, we get the rest walks from s_i by calling GET-MORE-WALK procedure.

THEOREM 3.3. *Algorithm MANY-RANDOM-WALKS (cf. Algorithm 5) solves k -RW-DoS for any $k \geq \ell^2$ and, with high probability, finishes in $O(\sqrt{\frac{k\ell \log n}{\delta}})$ rounds.*

PROOF. We claim that the algorithm finishes in $O(\frac{\eta\ell \log n}{\delta} + k/\eta)$ rounds with high probability. The theorem follows immediately using $\eta = \sqrt{\frac{k\delta}{\ell \log n}}$. Now we prove our claim.

It follows from Lemma 2.2 that Phase 1 finishes in $\frac{\eta\ell \log n}{\delta}$ rounds with high probability. Observe that the procedure GET-MORE-WALK is called at most k/η times and each time it uses at most ℓ rounds by Lemma 2.4. \square

Algorithm 5 MANY-RANDOM-WALKS($(s_1, k_1), (s_2, k_2), \dots, (s_q, k_q) \ell$)

Input: Desired walk length ℓ , q distinct sources s_1, \dots, s_q and number of walks k_i for each source s_i where $\sum_{i=1}^q k_i = k$

Output: For each i , destinations of k walks of length ℓ starting from s_i .

Phase 1: Each node performs η random walks of length ℓ . See phase 1 of SINGLE-RANDOM-WALK (cf. Algorithm 1).

Phase 2:

- 1: **for** $i = 1$ to q **do**
 - 2: **if** $k_i > \eta$ **then**
 - 3: s_i calls GET-MORE-WALK($s_i, k_i - \eta, \ell$).
 - 4: **end if**
 - 5: **end for**
-

4. K WALKS WHERE SOURCES OUTPUT DESTINATIONS (K -RW-SOD)

In this section we extend our results to k -RW-SoD using the following lemma.

LEMMA 4.1. *Given an algorithm that solves k -RW-DoS in $O(S)$ rounds, for any S , one can extend the algorithm to solve k -RW-SoD in $O(S + k + D)$ rounds.*

The idea of the above lemma is to construct a BFS tree and have each destination node send its ID to the corresponding source via the root. By using upcast and downcast algorithms [32], this can be done in $O(k + D)$ rounds.

PROOF. Let the algorithm that solves k -RW-DoS perform one walk each from source nodes s_1, s_2, \dots, s_k . Let the destinations that output these sources be d_1, d_2, \dots, d_k respectively. This means that for each $1 \leq i \leq k$, node d_i has the ID of source s_i . To prove the lemma, we need a way for each d_i to communicate its own ID to s_i respectively, in $O(k + D)$ rounds. The simplest way to do this is for each node ID pair (d_i, s_i) to be communicated to some fixed node r , and then for r to communicate this information to the sources s_i . This is done by r constructing a BFS tree rooted at itself. This step takes $O(D)$ rounds. Now, each destination d_i sends its pair (d_i, s_i) up this tree to the root r . This can be done in $O(D + k)$ rounds using an upcast algorithm [32]. Node r then uses the same BFS tree to route back the pairs to the appropriate sources. This again takes $O(D + k)$ rounds using a downcast algorithm [32]. \square

THEOREM 4.2. *Given a set of k sources, one can perform k -RW-SoD after random walks of length ℓ in*

$$O\left(\frac{(k\ell)^{2/3} D^{1/3} (\log n)^{1/3}}{\delta^{1/3}} + D\right) \text{ rounds when } k \leq \ell^2, \text{ and in } O\left(\sqrt{\frac{k\ell \log n}{\delta}} + k + D\right) \text{ rounds when } k \geq \ell^2.$$

PROOF. On applying Lemma 4.1 to Theorems 3.2 and 3.3, we get that k -RW-SoD can be done in $O\left(\frac{(k\ell)^{2/3} D^{1/3} (\log n)^{1/3}}{\delta^{1/3}} + k + D\right)$ rounds when $k \leq \ell^2$ and $O\left(\sqrt{\frac{k\ell \log n}{\delta}} + k + D\right)$ rounds when $k \geq \ell^2$. Note that when $k \leq \ell^2$, $(k\ell)^{2/3} \geq k$. \square

We show an almost matching lower bound below.

LEMMA 4.3. *For every $D \leq n$ and every k , there exists a graph G of diameter D such that any distributed algorithm*

that solves k -RW-SoD on G requires with high probability $\Omega(\min(\ell, D) + k)$ rounds.

PROOF. Consider a star graph of k branches and each branch has length ℓ . The lower bound of $\Omega(\min(\ell, D))$ rounds can be proved the same way as in Lemma 3.1 (by looking at the center node and any leaf node).

For the lower bound of $\Omega(k)$ rounds, let s be any neighbor of the center node and consider computing k walks of length 2 from s . With positive probability, there are $\Omega(k)$ different destinations. For s to know (and output) all these destination IDs, the algorithm needs $\Omega(k)$ rounds as the degree of s is just two. \square

5. BETTER BOUND WHEN $\ell \geq T_{MIX}$

In this section, we study the case when the length of the random walk sample is larger than the mixing time of the graph. This case is especially interesting in graphs with small mixing time, such as expanders and hypercubes (mixing time being $\tilde{O}(1)$). We show that for such graphs, random walks (both k -RW-DoS and k -RW-SoD) of length ℓ can be done more efficiently. In fact, we show a more general result: Random walks of length $\ell \geq t_{mix}$, where t_{mix} is the mixing time (the number of steps needed to be “close” to the stationary distribution, assuming that the graph is connected and non-bipartite) of an undirected unweighted graph G can be done in $O(D + k)$ rounds. Since mixing time depends on how close one wishes to be to the steady state distribution (there are various notions of mixing time, see e.g., [11]), we consider *approximate* sampling for this case. By this, we mean that we will sample nodes according to the steady state distribution which is *close* to the distribution after a walk of length ℓ when $\ell \geq t_{mix}$.

If one disregards the fact that this is an approximate sampling from the random walk, this improves the FEW-RANDOM-WALKS algorithm (cf. Algorithm 4) in this special case. Note that in the rest of the paper, we obtained *exact* random walk samples from the distribution of the corresponding length.

The rest of the section shows how one can sample a node according to the steady state distribution efficiently.

5.1 Algorithm

Our algorithm relies on the following observation.

Observation: *Approximately* sampling a node after a walk of length $\ell \geq t_{mix}$ only requires sampling a node v of degree d_v with probability $d_v/(2m)$.

This is due to the well-known fact that the distribution after a walk of length greater than the mixing time for undirected unweighted graphs is determined by the graph’s degree distribution. In particular, the stationary probability of a node of degree d is $d/2m$ where m is the number of edges in the graph [29].

We now only need to show that sampling such a node can be done in $O(D + k)$ rounds. In fact, one can show something more general. One can sample a node from any fixed distribution in $O(D + k)$ rounds. We present the algorithm for this in SAMPLE-FIXED-DISTRIBUTION(s, H) (cf. Algorithm 6). The algorithm can be thought of as a modification of the first two sweeps of algorithm SAMPLE-DESTINATION (cf. Algorithm 3); this extension does not require any additional insight, and so we have placed the algorithm in the appendix. The notation in the algorithm uses H to denote

Algorithm 6 SAMPLE-FIXED-DISTRIBUTION
 $(\{s_1, s_2, \dots, s_k\}, k, H)$

Input: Source nodes s_1, s_2, \dots, s_k , number of destinations to be sampled, k , and the distribution to be sampled from, H .

Output: k sampled destinations according to the distribution H .

- 1: Let r be any node (can be chosen by a leader election algorithm). Construct a BFS tree rooted at r . While constructing, every node stores its parent node/edge.
 - 2: Every source node sends their IDs to r via the BFS tree. r now samples k destinations as follows.
 - 3: k tokens are passed from leaves of the BFS eventually to r . The BFS is divided into D levels. Initially, each leaf node (level D node) fills all the k slots with its node ID and sends it to its parent. It also sends a value to reflect the sum of probabilities of all nodes in the subtree rooted at itself, according to the distribution H . For a leaf node, the value is its own probability in H .
 - 4: **for** $x = D - 1$ to 1 **do**
 - 5: When a node v in level x receives one or more sets of tokens and values from its children, it does the following for each token slot. For the first slot, suppose that it receives node IDs a_1, a_2, \dots, a_j with values p_1, p_2, \dots, p_j . Let a_0 denote its own ID and p_0 denote its distribution according to H . The node v samples one of a_0 through a_j , with probabilities proportional to the values; i.e., for any $i \leq j$, the node a_i is picked with probability $p_i / (p_0 + p_1 + \dots + p_j)$.
 - 6: The above is done for each of the k slots. The node v then updates the value to $p_1 + p_2 + \dots + p_j + p_0$. These k slots and the value are then sent by v to its parent (unless v is the root).
 - 7: **end for**
 - 8: Finally, r receives all destinations (IDs in the k slots). It randomly matches destinations with the sources. It then sends each destination ID to the corresponding source.
-

the distribution from which a node is to be sampled. Therefore, H_v is the probability with which the resulting node should be v . We state the algorithm more generally, where k nodes need to be sampled according to distribution H and the source needs to recover their IDs.

5.2 Analysis

We state the main result of this section below. The proof requires two parts - to verify the correctness of the sampling algorithm; and to bound the number of rounds.

THEOREM 5.1. *Algorithm SAMPLE-FIXED-DISTRIBUTION (cf. Algorithm 6) solves k -RW-DoS and k -RW-SoD in $O(D+k)$ rounds for any $\ell \geq t_{mix}$.*

PROOF. We first show that the nodes reaching the root are sampled from the true probability distribution H .

Consider any node b and let T be the subtree rooted at b . We claim that, for any node v in T , the probability v being in the first slot at b is exactly equal to $\frac{H_v}{H(b)}$ where $H(b) = \sum_{v \in T} H(v)$. We show this by induction. Our claim clearly holds for the base case where b is a leaf node. Now, for any non-leaf node b , assume that our claim is true for any children of b . That is, if b has j children and if a_i (for $1 \leq i \leq j$) is the node b receives from the i -th children,

denoted by u_i , then a_i is picked from u_i with probability $\frac{H_{a_i}}{H(u_i)}$. Note from the algorithm that $p_0 = 1$ and $p_i = H(u_i)$ for any $1 \leq i \leq j$. Therefore, for any i , the probability that a_i is picked by b is $\frac{H_{a_i}}{H(u_i)} \cdot \frac{H(u_i)}{H(b)} = \frac{H_{a_i}}{H(b)}$ as claimed.

Now, using the claim above, the probability of a node v being in the first slot of the root node is $\frac{H_v}{H(r)}$ where r is the root of the BFS tree of the entire graph. Since $H(r) = 1$, we have a sample from the required probability. This completes the proof of correctness.

Now we show that the number of rounds is $O(D+k)$. Constructing the BFS tree requires $O(D)$ rounds. The backward phase would require $O(D)$ rounds if only one token was being maintained, since the depth of the tree is $O(D)$. Now, with k tokens, observe that when a node receives tokens from all of its children it can immediately sample one token and forward it to its parent. In other words, it always sends messages to its parents once it receives messages from its children. Since each node receives only k sets of messages, the number of rounds is $O(D+k)$. \square

6. USING ROUTING TABLES

We make a remark about the use of GET-MORE-WALKS and SAMPLE-DESTINATION in our main algorithms including SINGLE-RANDOM-WALK. In GET-MORE-WALKS, we perform η walks of length λ from a specified source, in λ rounds. At the end of this, all destinations are aware of the source, however, the source does not know the destinations ids. This is why SINGLE-RANDOM-WALK needs to invoke SAMPLE-DESTINATION later on. An alternative to this is to invest more rounds in GET-MORE-WALKS as follows. The source can obtain all η destinations in $O(\eta + \lambda)$ rounds; this can be shown by a standard congestion + dilation argument. If this is done, then SAMPLE-DESTINATION is no longer required. The crucial point is that in the choice of our parameters, λ is more than η , and so the overall asymptotic bounds of the algorithm are not affected. Notice that we still need lD/λ rounds for the stitching phase (although we no longer need this for the SAMPLE-DESTINATION calls).

The advantage of getting rid of SAMPLE-DESTINATION is that the source node now does not need to construct BFS trees to obtain and sample from its destinations. If the nodes had access to a shortest path routing table, then the algorithm SINGLE-RANDOM-WALK will never need to construct BFS trees. While the construction of BFS tree in our algorithm is not a bottleneck in terms of number of rounds, this procedure is often unwieldy in practice. The use of routing tables instead, and simplifying the algorithm to not use SAMPLE-DESTINATION greatly increases the practicality of our method: no BFS tree construction is needed and the algorithm is very local and robust. Notice that the general idea of SAMPLE-DESTINATION is still required for our later results that use SAMPLE-FIXED-DISTRIBUTION.

7. CONCLUSION

To conclude, our main result is a $\tilde{O}(\ell^{2/3} D^{1/3})$ algorithm for 1-RW-DoS which is later extended to k -RW-DoS algorithms (using $\tilde{O}((k\ell)^{2/3} D^{1/3})$ rounds for $k \leq \ell^2$ and $\tilde{O}(\sqrt{k\ell})$ rounds for $k \geq \ell^2$). We also consider other variations, special cases, and lower bounds.

This problem is still open, for both lower and upper bound. In particular, we conjecture that the true number of rounds for 1-RW-DoS is $\tilde{O}(\sqrt{\ell D})$. It will be also interesting to ex-

plore fast algorithms for performing random walks in directed graphs (both weighted and unweighted).

As noted earlier, the focus of this paper is to improve the time complexity of random walks; however, this can come at the cost of increased message complexity. It would also be interesting to study tradeoffs between time and messages.

Acknowledgements. The first author would like to thank Gagan Goel for an initial discussion.

8. REFERENCES

- [1] L. A. Adamic, R. M. Lukose, A. R. Puniyani, and B. A. Huberman. Search in power-law networks. *Physical Review*, 64, 2001.
- [2] 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 *FOCS*, pages 218–223, 1979.
- [3] 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.
- [4] B. Awerbuch, A. Baratz, and D. Peleg. Cost-sensitive analysis of communication protocols. In *9th ACM PODC*, pages 177–187, 1990.
- [5] H. Baala, O. Flauzac, J. Gaber, M. Bui, and T. El-Ghazawi. A self-stabilizing distributed algorithm for spanning tree construction in wireless ad hoc networks. *J. Parallel Distrib. Comput.*, 63(1):97–104, 2003.
- [6] J. Bar-Ilan and D. Zernik. Random leaders and random spanning trees. In *Proceedings of the 3rd International Workshop on Distributed Algorithms*, pages 1–12, London, UK, 1989. Springer-Verlag.
- [7] T. Bernard, A. Bui, and O. Flauzac. Random distributed self-stabilizing structures maintenance. In *ISSADS*, pages 231–240, 2004.
- [8] A. R. Bharambe, M. Agrawal, and S. Seshan. Mercury: supporting scalable multi-attribute range queries. In *SIGCOMM*, pages 353–366, 2004.
- [9] A. Z. Broder. Generating random spanning trees. In *FOCS*, pages 442–447, 1989.
- [10] M. Bui, T. Bernard, D. Sohler, and A. Bui. Random walks in distributed computing: A survey. In *IICS*, pages 1–14, 2004.
- [11] F. Chung. *Spectral Graph Theory*. American Mathematical Society, Providence, RI, USA, 1997.
- [12] B. F. Cooper. Quickly routing searches without having to move content. In *IPTPS*, pages 163–172, 2005.
- [13] D. Coppersmith, P. Tetali, and P. Winkler. Collisions among random walks on a graph. *SIAM J. Discret. Math.*, 6(3):363–374, 1993.
- [14] A. Das Sarma, S. Gollapudi, and R. Panigrahy. Estimating pagerank on graph streams. In *PODS*, pages 69–78, 2008.
- [15] S. Dolev, E. Schiller, and J. L. Welch. Random walk for self-stabilizing group communication in ad hoc networks. *IEEE Trans. Mob. Comput.*, 5(7):893–905, 2006. also in *PODC’02*.
- [16] S. Dolev and N. Tzachar. Spanders: Distributed spanning expanders. *Dept. of Computer Science, Ben-Gurion University, TR-08-02*, 2007.
- [17] A. J. Ganesh, A.-M. Kermarrec, and L. Massoulié. Peer-to-peer membership management for gossip-based protocols. *IEEE Trans. Comput.*, 52(2):139–149, 2003.
- [18] C. Gkantsidis, M. Mihail, and A. Saberi. Hybrid search schemes for unstructured peer-to-peer networks. In *INFOCOM*, pages 1526–1537, 2005.
- [19] N. Goyal, L. Rademacher, and S. Vempala. Expanders via random spanning trees. In *SODA*, pages 576–585, 2009.
- [20] W. K. Hastings. Monte carlo sampling methods using markov chains and their applications. *Biometrika*, 57(1):97–109, April 1970.
- [21] A. Israeli and M. Jalfon. Token management schemes and random walks yield self-stabilizing mutual exclusion. In *PODC*, pages 119–131, 1990.
- [22] D. R. Karger and M. Ruhl. Simple efficient load balancing algorithms for peer-to-peer systems. In *SPAA*, pages 36–43, 2004.
- [23] D. Kempe, J. M. Kleinberg, and A. J. Demers. Spatial gossip and resource location protocols. In *STOC*, pages 163–172, 2001.
- [24] J. M. Kleinberg. The small-world phenomenon: an algorithm perspective. In *STOC*, pages 163–170, 2000.
- [25] C. Law and K.-Y. Siu. Distributed construction of random expander networks. In *INFOCOM*, 2003.
- [26] D. Loguinov, A. Kumar, V. Rai, and S. Ganesh. Graph-theoretic analysis of structured peer-to-peer systems: routing distances and fault resilience. In *SIGCOMM*, pages 395–406, 2003.
- [27] Q. Lv, P. Cao, E. Cohen, K. Li, and S. Shenker. Search and replication in unstructured peer-to-peer networks. In *ICS*, pages 84–95, 2002.
- [28] N. Metropolis, A. W. Rosenbluth, M. N. Rosenbluth, A. H. Teller, and E. Teller. Equation of state calculations by fast computing machines. *The Journal of Chemical Physics*, 21(6):1087–1092, 1953.
- [29] M. Mitzenmacher and E. Upfal. *Probability and Computing: Randomized Algorithms and Probabilistic Analysis*. Cambridge University Press, New York, NY, USA, 2005.
- [30] R. Morales and I. Gupta. Avmon: Optimal and scalable discovery of consistent availability monitoring overlays for distributed systems. In *ICDCS*, page 55, 2007.
- [31] G. Pandurangan and M. Khan. Theory of communication networks. In *Algorithms and Theory of Computation Handbook, Second Edition*. CRC Press, 2009.
- [32] D. Peleg. *Distributed computing: a locality-sensitive approach*. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 2000.
- [33] R. Sami and A. Twigg. Lower bounds for distributed markov chain problems. *CoRR*, abs/0810.5263, 2008.
- [34] D. B. Wilson. Generating random spanning trees more quickly than the cover time. In *STOC*, pages 296–303, 1996.
- [35] M. Zhong and K. Shen. Random walk based node sampling in self-organizing networks. *Operating Systems Review*, 40(3):49–55, 2006.
- [36] M. Zhong, K. Shen, and J. I. Seiferas. Non-uniform random membership management in peer-to-peer networks. In *INFOCOM*, pages 1151–1161, 2005.