Distributed Branching Random Walks and Their Applications

Vijeth Aradhya
National University of Singapore, Singapore

Seth Gilbert
National University of Singapore, Singapore

Thorsten Götte
University of Hamburg, Germany

Abstract

In recent years, the explosion of big data and analytics has necessitated distributed storage and processing with several compute nodes (e.g., multiple datacenters). These nodes collaboratively perform parallel computation, where the data is typically partitioned across these nodes to ensure scalability, redundancy and load-balancing. But the nodes may not always be co-located; in many cases, they are part of a larger communication network. Since those nodes only need to communicate among themselves, a key challenge is to design efficient *routes* catered to that *subnetwork*.

In this work, we initiate the study of distributed sampling and routing problems for subnetworks in any well-connected network. Given any network $G = (V, E)$ with mixing time τ_{mix}, consider the canonical problem of permutation routing [Ghaffari, Kuhn and Su, PODC 2017] that aims to minimize both congestion and dilation of the routes, where the demands (i.e., set of source-terminal pairs) are such that each node sends or receives number of messages proportional to its degree. We show that the permutation routing problem, when demands are restricted to any subset $S \subseteq V$ (i.e., subnetwork), can be solved in $\exp(O(\sqrt{\log |S|})) \cdot \tilde{O}(\tau_{\text{mix}})$ rounds (where $\tilde{O}(\cdot)$ hides polylogarithmic factors of $|V|$). This means that the running time depends *subpolynomially* on the subnetwork size (i.e., not on the entire network size). The ability to solve permutation routing efficiently immediately implies that a large class of parallel algorithms can be simulated efficiently on the subnetwork.

As a prerequisite to constructing efficient routes, we design and analyze *distributed branching random walks* that distribute tokens started by the nodes in the subnetwork. At a high-level, these algorithms operate by always moving each token according to a (lazy) simple random walk, but also branching a token into multiple tokens at some specified intervals; ultimately, if a node starts a branching walk, with its id in a token, then by the end of execution, several tokens with its id would be randomly distributed among the nodes. As these random walks can be started by many nodes, a crucial challenge is to ensure *low-congestion*, which is a primary focus of this paper.

2012 ACM Subject Classification Theory of computation → Distributed algorithms; Theory of computation → Random walks and Markov chains; Networks → Packet scheduling

Keywords and phrases Distributed Graph Algorithms, Random Walks, Permutation Routing

Digital Object Identifier 10.4230/LIPIcs.OPODIS.2024.36

Funding V. Aradhya and S. Gilbert acknowledge the support by Singapore MOE Tier 2 grant MOE-T2EP20122-0014. T. Götte acknowledges the support by DFG grant 491453517.

1 Introduction

When designing distributed algorithms, we typically develop solutions for an entire network. For example, we might want to identify a maximal independent set (MIS) for the graph (e.g., [28]), find an MST for the graph (e.g., [60]), compute all-pairs shortest paths (e.g., [42]), or solve a load balancing problem (e.g., [25]).

In practice, however, the "entire network" is gigantic, and the application we are designing likely wants to solve the problem only on a subnetwork. For example, we might need an MIS only for a specific subgraph – an easy problem, since finding an MIS depends only on the local neighborhood. Or we might want to find an MST connecting a subset of the nodes – i.e., we want a Steiner tree; over the last several years, there has been significant progress in distributed algorithms for Steiner tree approximation [50, 65].

For many other (non-local) problems, it remains open (to date) how easy or hard they are to solve on subgraphs. In this paper, we focus on two fundamental problems – sampling and routing. Solving these problems provides a framework for easily adapting existing algorithms designed for a graph G to run efficiently on a subgraph of G: by (effectively) building on overlay allowing nodes in the relevant subset to communicate efficiently, we can easily simulate global algorithms on a subgraph.

In more detail, imagine you are given some graph $G = (V, E)$ representing the network, and some subset $S \subseteq V$ of the nodes are identified as participants. The subgraph induced by S is not necessarily connected (without using edges from the larger graph). Thus to solve problems for the nodes in S, we need to design algorithms that operate in the larger graph – but that are ideally more efficient than simply solving the problem on the entire graph. We focus on networks with limited bandwidth – i.e., communication is governed by the CONGEST model [61]. And we assume the graph is reasonably well-connected, i.e., has a given mixing time τ_{mix}.

We consider two key problems: (i) random sampling, where every node in S (simultaneously) retrieves a random sample of node identifiers from the set S; and (ii) permutation routing, where nodes in S discover efficient (low congestion, low dilation) routes between pairs of nodes in S.

In general, the challenging aspect of solving these problems on a subgraph $S \subseteq V$ is establishing efficient communication between the nodes in S, without flooding the broader network with messages and creating too much congestion. Nodes in S know that they are in S, but they do not know anything about which other nodes are in S or where to find them. To create connections, these nodes in S might send out large numbers of messages to probe the network – but (depending on the topology of the network) this might induce significant congestion if done naively.

Distributed Sampling

More specifically, the goal of distributed sampling is that each node in S receives sufficiently many samples (i.e., node identifiers) drawn from a distribution sufficiently close to the stationary distribution of the graph.

▶ **Definition 1.** `Distributed-Subsample`(S) or `DSubsample`(S) problem.

- **Input:** *Network $G = (V, E)$; Subset of nodes $S \subseteq V$; $n = |V|$.*
- **Ensure:** *Each node v outputs $\Theta(d(v) \log n)$ "almost-random" samples (i.e., node ids) from set S, i.e., for each sample s at any node $v \in V$, for any node $w \in S$, $\Pr[s = w] = (d(w)/d(S)) \pm n^{-\Theta(1)}$, where $d(w)$ is the degree of node w and $d(S) = \sum_{u \in S} d(u)$.*

Random-walk based algorithms have been often used for sampling: random walks are fast and light weight, and this sampling capacity has been useful in a variety of contexts, for example, aggregate statistics (e.g., [53, 67]), resource discovery (e.g., [39, 38]), information dissemination (e.g., [33, 34]), peer-to-peer networks (e.g., [47, 37, 6, 35]), agreement and leader election (e.g., [4, 5, 36]), etc.

Unfortunately, simple random walks may take too long to discover and sample from a subset of nodes S in the graph.

Branching Random Walks

The key technical tool that we develop to solve this problem is a special type of branching random walk that we call an *amoeba walk*. Branching random walks allow a token to split during the walk, producing more tokens and covering the graph faster. Thus a branching walk will more quickly discover a hidden subset of the graph.

The key challenge with branching walks is controlling congestion: as the number of tokens multiplies, so does (potentially) the congestion. In fact, we show (in Theorem 5) that a simple branching random walk (i.e., a "count random walk" [45, 36]) inevitably creates too much congestion.

Many branching random walks – like cobra walks (coalescing and branching random walks) [22, 54] – control congestion by coalescing tokens when there are too many. They are generally designed to begin at one location and eventually "cover" the network with tokens representing a source message (e.g., a rumor or an infection).[1]

Unlike such branching walks, we want to begin random walks at many different nodes in the network, where each source achieves some "balanced" level of cover in the network. (For example, we do not want one source to crowd out the others.) To ensure this, we do not want to allow arbitrary branching at every step – we want a more controlled process.

Specifically, an amoeba walk alternates between two different types of phases: mixing and dividing. During the mixing phase, a token follows a traditional random walk (without any branching); during the dividing phase, it splits into multiple tokens. This alternation prevents too much congestion from developing in any one place in the network. Thus, if each node in S starts appropriate amoeba walks, tokens from each node will eventually be randomly distributed in the graph, providing a random sample of other nodes in S.

Distributed Routing

Building on the sampling infrastructure (which connects each node in S to a small number of other nodes in S), we show how to develop a routing algorithm among all the nodes in S, allowing nodes in S to communicate efficiently.

Specifically, we focus on the distributed permutation routing problem, introduced by Ghaffari, Kuhn and Su [31] in the context of expander networks. In this problem, there is a set of source-terminal pair of nodes (typically called the "demand"), where a (unique) message needs to be sent from each source to its terminal. At the same time, there is a constraint on the demand wherein each node is designated as the source or terminal for a total number of messages that is at most its degree.

The challenge is to ensure that all messages reach their respective destinations as quickly as possible, requiring nodes to route them all in parallel, while limiting congestion, i.e., ensuring that at most *polylogarithmic* (in network size) bits are sent along any edge at a time.

This problem has received great attention in recent years [31, 32, 14, 12], due to several important applications, such as distributed MST and min-cut [31, 16], data summarization (e.g., sorting, distinct elements, frequency moments, etc) [67], subgraph enumeration [13],

[1] Of course, one can start a cobra walk at many locations, and there are other applications aside from "rumor spreading."

shortest path and distance-related computation [11], etc. Moreover, Ghaffari and Li [32] showed that permutation routing enables an efficient adaption of work-efficient algorithms in the PRAM model to distributed networks in the CONGEST model, thereby creating a bridge to a vast literature of parallel algorithms.

In this work, we study a natural generalization of permutation routing, where the demand is restricted to a *subset* of nodes. The goal is to obtain performance guarantees that (ideally) depend on the subset size.

▶ **Definition 2.** `Subnetwork-Permutation-Routing`(S) or `SPRoute`(S) problem.

- **Input:** *Network* $G = (V, E)$*; Source-Terminal Set* $\mathcal{D} = \{(s_1, t_1), \dots\}$ *where* $\forall i \in [|\mathcal{D}|], \{s_i, t_i\} \subseteq S \subseteq V$ *and node* s_i *knows the id of node* t_i *and a message* m_i *that needs to be sent to node* t_i*, where* m_i *is of size at most* $O(\log |V|)$ *bits.*
- **Require:** *For each node* $v \in S, |\{P \mid (v \in P) \wedge (P \in \mathcal{D})\}| \leq d(v) \cdot O(\log |V|)$*, where* $d(v)$ *is the degree of node* v*.*
- **Ensure:** $\forall i \in [|\mathcal{D}|]$*, node* t_i *receives the intended message* m_i*.*

Our basic solution for subnetworks extends the random-walk based approach of Ghaffari, Kuhn and Su [31]. First, the nodes execute (slightly modified) amoeba walks to randomly distribute the node ids. Then the nodes in S efficiently simulate a random graph overlay among themselves. Combining these methods, we can create a hierarchical decomposition of the nodes in S, enabling an efficient routing algorithm between any pair of those nodes.

Contributions

Our technical contributions are two-fold, with a focus on distributed sampling and routing.

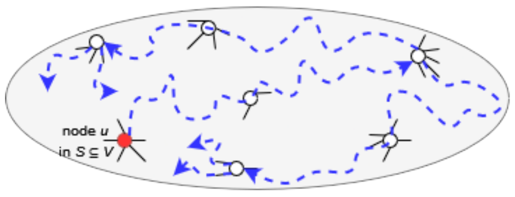

Figure 1 An illustration of amoeba-walk started by a node.

To tackle the sampling and routing problems, we design and analyze a new distributed branching random walk algorithm called *amoeba-walks*. The main idea is that each node in the subnetwork, starts a token that always executes a (lazy) simple random walk, but also occasionally branches into multiple tokens. See Figure 1 for an illustration. The tokens executing these algorithms carefully alternate between "mixing" and "dividing" phases, so that the algorithm induces low-congestion, and all the nodes end up with (almost-)random samples (i.e., node ids) from the subnetwork.

▶ **Theorem 3.** *Consider any network* $G = (V, E)$ *with mixing time* τ_{mix} *and any set* $S \subseteq V$*. The* `DSubsample`(S) *problem can be solved in* $\tilde{O}(\tau_{\text{mix}})$ *rounds whp.*

Once we are able to efficiently distribute the random samples of the subnetwork, one may be tempted to let the nodes simply reverse the tokens to obtain low-congestion routes. However, for the permutation routing problem, the demands can be such that each node sends or receives a number of messages *proportional* to its degree, which could cause congestion in the (naive) amoeba-walk algorithm. However, with a slight modification, and simulation of a (virtual) random graph over the subnetwork, and leveraging the state-of-the-art permutation routing algorithm (i.e., [32]), we prove the following theorem.

▶ **Theorem 4.** *Consider any network $G = (V, E)$ with mixing time τ_{mix} and any set $S \subseteq V$. The* `SPRoute`(S) *problem can be solved in* $\exp(O(\sqrt{\log |S|})) \cdot \tilde{O}(\tau_{\text{mix}})$ *rounds whp.*

An immediate corollary of this theorem is that we can simulate a large number of parallel and distributed algorithms designed for an entire graph [2, 32]. For example, due to Ghaffari and Li [32], Theorem 4 implies that any (work-efficient) T round CRCW PRAM algorithm can be simulated by the nodes in S in time $O(T \exp(O(\sqrt{\log |S|})) \cdot \tilde{O}(\tau_{\text{mix}})$.

2 Preliminaries

We formally define the distributed network model, assumptions, and notations.

Network. Let $G = (V, E)$ be a connected graph that denotes a network of nodes, where V and E are the set of nodes and set of (bidirectional) communication links, respectively. As is typical, $|V| = n$ and $|E| = m$. The *nodes* in the graph represent a fixed set of computing entities, where each node is associated with a unique *identifier* (or id). Moreover, we assume that the id of any node can be encoded in $O(\log n)$ bits.

The system proceeds in *synchronous rounds*, i.e., a message m sent by a node u in any round is received by its recipient node v by the end of that round.

Network Properties. Let $d(u) = |N(u)|$ and $d(S) = \sum_{u \in S} d(u)$ be the *degree* of node u and *volume* of subset $S \subseteq V$ (resp.), where $N(u) = \{v \mid (u, v) \in E\}$ is the set of *neighbors* of node u. Let $\Delta = \max_{u \in V} d(u)$ be the *max-degree*. Let $\Phi = \min_{S \subset V} |\partial S| / \min(d(S), d(V \setminus S))$ denote the conductance where $\partial S = \{(u, v) \mid u \in S, v \in V \setminus S\}$.

Let π and τ_{mix} denote the *stationary distribution* and *mixing time* of the lazy random walk on the network. Recall that the lazy walk stays at the current node with probability $1/2$, otherwise it moves to a uniformly random neighbor; see, Algorithm 2 for a reference. As the name implies, once the random walk reaches the "stationary" distribution, it stays there forever; for the lazy walk, it is known that $\pi(u) = d(u)/2m$ for any node u (see, e.g., [51]).

Furthermore, let $p_t(u, v)$ be the probability distribution of the lazy random walk that started at node u and ended up at node v after t steps. In this paper, we define the mixing time τ_{mix} as the minimum t such that for all pairs of nodes u and v, $|p_t(u, v) - \pi(u)| \leq n^{-c}$, where c is a suitable constant.

Congestion. In any round, each node can send $O(\log^2 n)$ bits along any edge. We consider a slightly non-standard CONGEST(B) model [61], where B is polylogarithmic in n (instead of $O(\log n)$), so that the exposition of the analysis becomes simple, as each node can send $O(\log n)$ random walk tokens, having distinct node ids, in a single round.

Estimates of Certain Global Parameters. For the sake of a clean exposition, we make a simplifying assumption that the nodes have a *good estimate* (i.e., within constant factors) of the network's mixing time, $d(S)$, n and m (where $S \subseteq V$ is part of the input).

For the routing problem, these parameters could be estimated directly, i.e., nodes can determine the various sizes by evaluating aggregate functions (such as summation, etc), for e.g., by constructing a BFS tree using the node with minimum id as the root. For the mixing time, nodes can use a "guess-and-double" trick until all routing paths are established [32], i.e., if a node cannot route a message to its destination in the anticipated time, then the node can inform the root, which can re-initiate the routing algorithm after doubling the estimate.

3 Related Work

Distributed and Parallel Random Walks. There has been a vast literature on the design, analysis and applications of random walks in distributed networks. Alon et al. showed that the cover time (i.e., expected time taken to visit all nodes) can be significantly reduced over various families of graphs, when multiple (independent) simple random walks are started from an arbitrary node. However, they do not minimize congestion. In the CONGEST model, Das Sarma et. al showed that an L-length random walk can be computed in $\tilde{O}(\sqrt{LD})$ time, i.e., sublinear in the length of the walk, where D denotes the diameter of the network. They achieved it by performing several short random walks and carefully stitching them together. In a subsequent work, Das Sarma et. al [66] extended this algorithm to dynamic regular networks, where the network can change arbitrarily over time. In fact, the idea of stitching independent random walks has been exploited in the massively parallel computation [46] and overlay models [21, 7], for an exponential speed up in computing random walks.

A closely related stochastic process is a natural generalization of simple random walk, called the coalescing-branching random walks [23, 19, 20, 55, 9]. Dutta et al. [23] initiated the study the rate of information dissemination, particularly cover time for different classes of networks. In particular, they show a cover time of $O(\log^2 n)$ in expander networks, whereas, the cover time for simple random walk is known to be $\Theta(n \log n)$. This process begins with a branching factor k, and an arbitrary node is initially "active". In any step, each active node chooses k random neighbors to be active (i.e., "branching" property). However, a node can be active in any step only if it is chosen by some node in the previous step (i.e., "coalescing" property). A key difference with our distributed branching random walk is that although the tokens divide into multiple tokens over time, they never coalesce into a *single* token at any node. Moreover, our focus is on ensuring that the process induces *low-congestion* (i.e., each node sends at most polylog bits over any edge in any round), whilst achieving a different goal, i.e., randomly distributing samples (i.e., node ids) of any subset of nodes to the network.

Perhaps, a type of distributed random walk that is closest to the distributed branching random walks, are "count" random walks [45]. Here, a small set of nodes start (polynomially) many walks by associating a count parameter with a token (in addition to the id of the node that started the token). The way that many random walks, by a single node, can be executed is by treating each count in each token as an independent random walk, but merging them (if they are associated with the same id) while sending them along any edge. This technique has proven to be useful in many important contexts (in the CONGEST model), such as approximating the mixing time [59], leader election with sublinear message bounds [45, 36], and testing of network conductance [26, 8]. An important observation in all those works, is that at most $\tilde{O}(1)$ number of nodes start the count random walks. Thus, regardless of the count values in the tokens, the number of tokens with *distinct* ids traversing the network is always low (due to merging of tokens). However, amoeba-walk tokens can be started by any (polynomial) number of nodes (with distinct ids), which makes the problem of maintaining low-congestion much harder to achieve, even if merging of tokens is allowed.

Expander Routing. While early solutions to permutation routing over expander networks were randomized [31, 32], recent works have also designed efficient deterministic algorithms [14, 12]. Since our solution uses, as a black-box, the permutation routing algorithm designed for the *entire* network, we provide a high-level overview of the algorithm.

The randomized algorithms work by recursively embedding Erdős-Rènyi random graphs $\mathcal{G}(n, p)$ (whilst deterministic solutions work with embedding an expander graph), on a base graph G_0, using random walks. In particular, the base graph G_0 is a random graph $\mathcal{G}(m, d/m)$

with a parameter $d = \exp(\Theta(\sqrt{\log n}))$, constructed over the edge-set [32]. Then, a graph G_1 is embedded onto G_0 with low congestion and dilation, where G_1 is a disjoint union of random graphs G_1^i's, where G_1^i is a random graph $\mathcal{G}(|V_i|, d/4|V_i|)$ over the set of nodes V_i, where $V_1, V_2, \ldots, V_k$ form a partition of V, and for all $i, |V_i| = \Theta(m/k)$. Such a hierarchical decomposition recursively proceeds until the graphs have a size of roughly $\exp(\Theta(\sqrt{\log n}))$. Finally, a routing mechanism is given using standard packet-routing techniques, i.e., randomized mapping of address space (e.g., [68, 43, 64]) and random-delay packet routing [48].

Connection to Flow Sparsifiers. The notion of cut and flow *vertex* sparsifiers were introduced in the seminal works of [57, 49]. Let $G = (V, E)$ be a network, and *terminal* set $K \subset V$, then informally, the problem is to construct a graph $H = (V_H, E_H)$ that preserves (i.e., bounded by a small multiplicative factor) the minimum congestion of flows for *any* demand that is *restricted* to the terminals. Note that for any given demand, there exists an optimal "flow assignment," i.e., a set of paths along which messages from a source to a terminal are routed. Here, the optimality is defined with respect to minimizing the congestion objective[2], which is defined as the maximum congestion over all edges, and the *congestion* of any edge is the total flow sent over that edge (divided by its capacity).

The main idea is that if a combinatorial optimization problem depends *only* on the flows (or cuts) of a *subset* of vertices, called terminals, then an algorithm for that problem can be run on H (after having *precomputed* such a sparsifier), instead of G, thereby significantly reducing the runtime, as $|V_H| \approx |K| \ll |V|$. Such a fundamental "network compression" primitive has received a great deal of attention since its inception (see, e.g., [24, 15, 52, 18, 1, 44, 17]).

In comparison with our work, we point out a few differences with flow sparsification. First and foremost, we consider identifying a *subgraph*, i.e., $V_H \subseteq V$ and $E_H \subseteq E$ in the *distributed* setting, where nodes have only local knowledge. Second, another key difference is that our focus is on minimizing the *completion-time* objective, i.e., the time taken for the last message to reach its destination, which is a function of *both* congestion and dilation (i.e., length of the paths). We refer to the recent works of [30, 40] for more details on this objective (and how it is strongly desirable in packet routing in networks). Finally, we consider preserving the minimum completion-time for permutation demands, and not arbitrary demands.

4 Distributed Branching Random Walks

In this section, we design and analyze a distributed random walk algorithm called *amoeba-walk*. Broadly, in this algorithm, the nodes send out tokens, where the tokens always execute simple random walk for a certain number of rounds (i.e., mix), and every so often, each of them branch into *two* tokens (i.e., divide). The *lazy* version of an amoeba-walk can be analogously defined, i.e, the tokens would always execute the lazy simple random walk. In this paper, we work with lazy random walks, as the simple random walk may not always converge to the stationary distribution, for e.g., in bipartite networks.

Each node u in subset $S \subseteq V$, starts the "amoeba-walk token," $t_u = (u.id, L, count_u)$, where the token consists of three entries: (1) $u.id$ is the id of node u (that started the token), (2) L is the number of steps after which the token divides into two tokens, and (3) $count_u$ denotes the "count" of the token t_u.

The count of an amoeba-walk token plays an important role because it represents the "weight" of the token; in other words, one can think of the token as a set of (lazy simple random walk) tokens moving *together*, with the cardinality of the set equal to the count.

[2] For formal definitions in vertex sparsification, we refer to [58].

Whenever a token divides into two tokens, its count is equally divided amongst those two tokens. However, when a token has count equal to 1, then that token will not divide into multiple tokens. Thus, finally, at the end of the execution, each amoeba-walk token has its count equal to 1. Since our goal is to (randomly) distribute the node ids of the subnetwork in the *entire* network, the total count of all amoeba-tokens at the start of the execution, is set such that $\sum_{u \in S} count_u = \Theta(m \log n)$. See Algorithm 1 for the pseudocode.

Algorithm 1 Amoeba-Walk(S, L) for a node u.

```
1:  Let $\alpha = \Theta(1)$ and $\mathcal{C} = \lceil \alpha m d(u) \log n / d(S) \rceil$ and token $t = (u.id, L, \mathcal{C})$;
2:  (Round 1) $\wedge$ $(u \in S) \rightarrow$ **LazyRW**$(t)$;
3:  **for** each received token $t = (id, l, count)$ **do**
4:      // Mix Phase
5:      $l \geq 1 \rightarrow$ **LazyRW**$((id, l-1, count))$;
6:      **if** $l < 1 \wedge count > 1$ **then**
7:          // Divide Phase
8:          **LazyRW**$((id, L, \lceil count/2 \rceil))$;
9:          **LazyRW**$((id, L, \lfloor count/2 \rfloor))$;
10:     **end if**
11: **end for**
```

Algorithm 2 LazyRW$(token)$ for a node u.

```
1: Let $b \in_R \{0, 1\}$ denote a (uniformly) random element drawn from the set $\{0, 1\}$;
2: $b = 0 \rightarrow$ Send $token$ to a random node in $N(u)$;
3: $b = 1 \rightarrow$ Send $token$ to self;
```

First, we provide a congestion analysis to *necessitate* the mixing of tokens, before dividing them, i.e., when the parameter $L = 0$, we show that the amoeba-walk algorithm can cause congestion issues at some edge (e.g., some node is not able to forward a token along an edge in a round). Moreover, this analysis also sheds light on the shortcomings of the existing count random walk algorithm (e.g., [45, 59, 26, 36, 8]; see Section B for a similar analysis), especially in the case where the total number of nodes starting the tokens is high. Subsequently, we prove that the amoeba-walk has low-congestion when L is equal to $\Theta(\tau_{\text{mix}})$.

▶ **Theorem 5.** *Consider a network $G = (V, E)$ with conductance $\Phi = \Omega(1)$ and max-degree $\Delta = O(1)$. There exists a subset of nodes $S \subset V$ for which at least one node sends more than $\Omega(\log^c n)$ bits in expectation, for any constant c, along an edge in a single round during the execution of* **Amoeba-Walk**$(S, 0)$.

Proof. If the set of nodes S are "close together" in the network, and if $|S|$ is large enough, when the tokens have a high count and are dividing into multiple tokens, there are nodes in S that end up getting a large number of tokens (from many distinct nodes). The key idea is to shift our focus from a collection of tokens to individual tokens of count value equal to one from round 1, i.e., *each* count is indeed executing a lazy random walk (though the random choice may be correlated with other counts).

Notation. Let the initial count (in round 1), started by nodes in S, be denoted as $\mathcal{C} \in \mathbb{N}$. For the sake of analysis, the initial count started by node in S, is viewed as a *collection* of individual counts, indexed by $[\mathcal{C}]$. Let $C_t(u, v)$ be the random variable for the total count of node u (over all tokens present) at any node $v \in V$ in any round $t \geq 1$. Let $C_t(u, v, k)$ be the indicator random variables for the presence of the count k of node $u \in S$ at any node $v \in V$ in any round $t \geq 1$. Let $\mathrm{P}_{i,j}(u,v)$ denote the probability that a lazy random walk that starts at node u in round i and ends at node v in round j, for $j > i$. Let $\mathrm{sgn}(x) = 0$ if x is even, and 1 otherwise. Let $B(u,r) = \{v \mid 0 \leq \mathit{dist}(u,v) \leq r\}$ be the ball of radius r around a node u, where $\mathit{dist}(u,v)$ denotes the distance between nodes u and v in G. Let $\epsilon = \Phi/\Delta$ and $h = \lceil \log_{1+\epsilon} 10\mathcal{B} \rceil$ where $\mathcal{B} = \Omega(\log^c n)$; note $\forall u \in V, |B(u,H)| \geq 10\mathcal{B}$.

Setting up the Bottleneck. Consider some node $v_0 \in V$ and $|S| = \lfloor \sqrt{n} \rfloor$. Let $r_0 = \mathrm{argmin}_r(|B(v_0,r)| \geq |S|)$. Let S be the set of nodes *closest* to the node v_0 among nodes in $B(v_0,r_0)$, including node v_0, where $|S| \geq \lfloor \sqrt{n} \rfloor$. Let $S_{even} = \{w \mid \mathrm{sgn}(\mathit{dist}(v_0,w)) = 0 \wedge (w \in S)\}$ and $S_{odd} = S \setminus S_{even}$. By pigeonhole principle, either $S_{even} \geq 5\mathcal{B}$ or $S_{odd} \geq 5\mathcal{B}$. If $S_{even} \geq 5\mathcal{B}$, then $S' = S_{even}$ and $b = 0$, whereas if $S_{odd} \geq 5\mathcal{B}$, then $S' = S_{odd}$ and $b = 1$. Furthermore, if $\mathrm{sgn}(h) = b$, then $h' = h$, whereas if $\mathrm{sgn}(h) \neq b$, then $h' = h - 1$.

Analysis. As each count is executing a lazy random walk, for any nodes $u \in S$ and $v \in V$, and any count $k \in [\mathcal{C}]$, $\mathbb{E}[C_t(u,v,k)] = \mathrm{P}_{1,t}(u,v) \geq 1/(2\Delta)^t$. By linearity of expectation, and due to max-degree Δ, the following two claims hold: (1) $\forall u \in S$, if $0 < \mathit{dist}(u,v_0) = t$, then $\mathbb{E}[C_t(u,v_0)] \geq \lfloor \mathcal{C}/(2\Delta)^t \rfloor$, and (2) $\forall u \in S$, if $0 \leq \mathit{dist}(u,v_0) \leq t \leq h-2$ and $\mathrm{sgn}(t) = \mathit{dist}(u,v_0)$, then $\mathbb{E}[C_{t+2}(u,v_0)] \geq \mathbb{E}[C_t(u,v)]/4\Delta^2$. By combining the two claims, and the bound on conductance, $\forall u \in S'$, $\mathbb{E}[C_{h'}(u,v_0)] \geq \lfloor \Theta(\sqrt{n}\log n)/(2\Delta)^{h'} \rfloor = \Omega(\log n)$, as $\Delta = O(1)$ and $\Phi = \Omega(1)$, $(2\Delta)^{\log_{1+\epsilon} 10\mathcal{B}} \ll \sqrt{n}$. Thus, the node v_0 needs to send tokens from $\mathcal{B} = \Omega(\log^c n)$ *distinct* nodes in round $h'+1$, in expectation. ◀

For showing that amoeba-walks can have low-congestion, we rely on a useful property about multiple lazy random walks. Informally, it says that if each node u starts with roughly $\Theta(d(u)\log n)$ random walk tokens (i.e., the *collective* set of tokens[3], in the network, are roughly already initially in stationary distribution), then in any round t, the expected number of tokens at any node u is at most $O(d(u)\log n)$; if $t \leq \mathrm{poly}(n)$, this property also holds, whp. Indeed, variants of this property appear in different contexts of distributed networks, including overlay networks (e.g., [6, 21]) and distributed property testing (e.g., [10]).

▷ Claim 6. Consider any network $G = (V,E)$. Let each node u start at most $2B_u$ tokens that independently execute lazy random walk (i.e., Algorithm 2) where $B_u = Cd(u)\log n$, where C is a suitably large constant. Then, in any round $t \in \tilde{O}(\tau_{\mathrm{mix}})$, the number of tokens at any node u is at most $O(B_u)$ with high probability.

Proof. For the sake of analysis, at the start, add "virtual" tokens to each node until each node u has $2B_u = \Theta(d(u)\log n)$ (real or virtual) tokens. Note that adding additional tokens can only increase the congestion. Without loss of generality, let the initial vector of tokens is $\mathcal{B} = (2B_{u_1}, \ldots, 2B_{u_n})$. Further, let T be the lazy random walk transition matrix of G; it is well-known that the stationary distribution vector is $\pi = (d(u_1)/2m, \ldots, d(u_n)/2m)$. Using the property of the stationary distribution, we know that $\pi \cdot T = \pi$. With the help of this fact, we can bound the expected number of tokens at any node.

Let X_u^t the number of tokens at node u in round t. Let $\mathcal{X}_t = \left(X_{u_1}^t, \ldots, X_{u_n}^t\right)$ denote the vector of tokens in round t. We can show the following statement via induction for any $t \geq 1$,

$$\left(\mathbb{E}[X_{u_1}^t \mid \mathcal{X}_1 = \mathcal{B}], \ldots, \mathbb{E}[X_{u_n}^t \mid \mathcal{X}_1 = \mathcal{B}]\right) = \mathcal{B}.$$

[3] Here, we mean the distribution of tokens, i.e., the number of tokens at any node divided by the total number of tokens in the network.

For the first round, the statement is trivially true. Moreover, from any round to the next, an elementary calculation reveals that,

$$\begin{aligned}
&\big(\mathbb{E}[X_{u_1}^{t+1} \mid \mathcal{X}_1 = \mathcal{B}], \ldots, \mathbb{E}[X_{u_n}^{t+1} \mid \mathcal{X}_1 = \mathcal{B}]\big) \\
&= \big(\mathbb{E}[X_{u_1}^{t} \mid \mathcal{X}_1 = \mathcal{B}], \ldots, \mathbb{E}[X_{u_n}^{t} \mid \mathcal{X}_1 = \mathcal{B}]\big) \cdot T \\
&= \mathcal{B} \cdot T.
\end{aligned}$$

Now we use the fact that $\mathcal{B}_0$ is proportional to the stationary distribution,

$$\begin{aligned}
\big(\mathbb{E}[X_{u_1}^{t+1} \mid \mathcal{X}_1 = \mathcal{B}], \ldots, \mathbb{E}[X_{u_n}^{t+1} \mid \mathcal{X}_1 = \mathcal{B}]\big) &= \mathcal{B} \cdot T \\
&= (2Cd(u_1)\log n, \ldots, 2Cd(u_n)\log n) \cdot T \\
&= 4C \cdot m \log n \cdot \left(\frac{d(u_1)}{2m}, \ldots, \frac{d(u_1)}{2m}\right) \cdot T \\
&= 4C \cdot m \log n \cdot \pi \cdot T \\
&= 4C \cdot m \log n \cdot \pi = \mathcal{B}
\end{aligned}$$

Thus, for any round t, $\mathbb{E}[X_u^t \mid \mathcal{X}_1 = \mathcal{B}] = 2B_u$. Since the mixing time of lazy random walk is at most polynomial in n [51], $t = \tilde{O}(\tau_{\text{mix}}) \leq \text{poly}(n)$. As X_u^t is a sum of binary independent random variables over all tokens on all nodes, by Chernoff bounds (cf. Section A) and union bound, the number of tokens at any node u in any round t is $O(d(u)\log n)$ whp. $\triangleleft$

To prove that amoeba-walks, for a parameter $L = \Theta(\tau_{\text{mix}})$, respects the congestion requirements, we exploit a careful alternation between the reliance on *individual* (i.e., for "divide" phase) and *collective* (i.e., for "mix" phase) probability distribution of tokens.

▶ **Lemma 7.** *Consider any network $G = (V, E)$ with mixing time τ_{mix} and any set $S \subseteq V$. The following statements hold for the execution of* **Amoeba-Walk**$(S, L = \Theta(\tau_{\text{mix}}))$*, whp.*

1. *Each node sends or receives $O(\log^2 n)$ bits in any round.*
2. *After $\Theta(L \log n)$ rounds, for every token $t = (id, l, c)$ on any node, $l = 0$ and $c = 1$.*
3. *After $\Theta(L \log n)$ rounds, the total number of tokens present at any node $v \in V$, amounts to $\Theta(d(v)\log n)$, where $d(v)$ is the degree of node $v \in V$ in round 1.*

Proof. We analyze the algorithm in $O(\log n)$ "epochs," where epoch $i \geq 1$ starts in round $1 + (i-1)L$, i.e., the length of each epoch is L rounds. We maintain two invariants in every epoch, whp; for every node $u \in V$, (1) at the start, node u has at most $B_u = \Theta(d(u)\log n)$ tokens, and (2) in any round, node u has $O(B_u)$ tokens. If this can be shown, note that the first two statements (in the lemma) are proved.

Weighted Balls-and-Bins. Consider a stochastic process where, at most $Cm \log n$ balls are randomly thrown into n bins, where C is a suitably large constant. Specifically, each ball is independently placed in bin u with probability $p(u) = (d(u)/2m) \pm n^{-\Theta(1)}$, where $\sum_v p(v) = 1$. Let $X_{i,u}$ denote the indicator random variable for ball i placed in bin u. Let $X_u = \sum_i X_{i,u}$ denote the random variable for number of balls in bin u. By a Chernoff bound (cf. Section A) and union bound over all balls and bins, $X_u \leq \Theta(d(u)\log n) = B_u$, whp, where B_u denotes the "max load" on any bin u.

Congestion Analysis for Lazy Random Walks. Let every node u start at most $2B_u$ tokens that independently execute the lazy random walk in any round $i \geq 1$ over the network G. As the tokens are *collectively* in stationary distribution, by Claim 6, node u has at most $O(d(u)\log n)$ tokens in any round, whp.

Analysis of Amoeba-Walks. At the start of any epoch i, after the token-division (i.e., Line 7–9 in Algorithm 1), if at every node u, there are at most $2B_u$ tokens, then by the congestion analysis of lazy random walks, until the start of epoch $i+1$, the number of tokens at any node u is at most $O(B_u)$, whp.

At the end of any epoch i, i.e., right before the next token-division in round $1+iL$, each token is (independent of other tokens) "well-mixed," as it traversed the network for $\Theta(\tau_{\text{mix}})$ rounds (i.e., from the start to end of epoch i). In other words, the probability of that token ending up at any node u at round $1+iL$ is $(d(u)/2m) \pm n^{-\Theta(1)}$ (cf. Section 2). Thus, we can recover the bound B_u on the number of tokens at any node u at the start of epoch $i+1$ by the aforementioned weighted balls-and-bins process, with high probability. Finally, by a union bound on all rounds, **Amoeba-Walk**$(S, \Theta(\tau_{\text{mix}}))$ acheives the congestion bounds in every round, with high probability.

Finally, as there are (totally) $\Theta(m \log n)$ tokens in final phase, with each of them having independently executed lazy random walk, the third statement holds, whp, again due to the weighted balls-and-bins process. ◀

5 Applications

In this section, we provide applications of the distributed branching walk algorithm to the aforementioned sampling and routing problems for subnetworks.

5.1 Subnetwork Sampling

▶ **Theorem 3.** *Consider any network $G = (V, E)$ with mixing time τ_{mix} and any set $S \subseteq V$. The* `DSubsample`(S) *problem can be solved in $\tilde{O}(\tau_{\text{mix}})$ rounds whp.*

Proof. First, the nodes execute **Amoeba-Walk**$(S, L = \Theta(\tau_{\text{mix}}))$ algorithm to randomly distribute the tokens in the network. Due to Lemma 7, after $O(L \log n)$ rounds, the total number of tokens (with count equal to 1) at any node $v \in V$, amounts to $\Theta(d(v) \log n)$ whp.

Forward Mixing. Recall that when the count value in any token is equal to 1 (i.e., after a token-division), the token (independently) executes lazy random walk for $\Theta(\tau_{\text{mix}})$ rounds, before halting at a node. Thus, the probability of that token ending up at any node v is $(d(v)/2m) \pm n^{-c(L)}$, where $c(L)$ is some large constant depending on the parameter L (cf. Section 2). However, at the end of execution, for such a token (with count equal to 1), we want to analyze the final probability distribution of the *source* of the token.

Backward Mixing. For the sampling analysis, the reversibility property of random walks (see, e.g., [3]) turns out to be useful. Consider any node $w \in S$ and any token (referred to as "sample") s (where s denotes the node id of the token) at any node $v \in V$. Let us define two events $\mathbf{A}$ and $\mathbf{B}$, where $\mathbf{A}$ refers to $s = w$ and $\mathbf{B}$ refers to $s \in S$. Using reversibility, we consider the sample s doing a lazy random walk, starting from the node v, backwards in time (up until round 1). Thus, similar mixing arguments apply to this sample, even in the reverse sequence, with following implications, whp,

$$\Pr[\mathbf{A} \cap \mathbf{B}] = \frac{d(w)}{2m} \pm \frac{1}{n^{c(L)}} \text{ and } \Pr[\mathtt{B}] = \frac{d(S)}{2m} \pm \frac{1}{n^{c(L)}}.$$

Using conditional probability (cf. Section A), and the following ratio manipulation, we get that $\Pr[\mathbf{A} \mid \mathbf{B}] = (d(w)/d(S)) \pm O(n^{-c(L)+2})$, with high probability. Here, we show the steps for the upper bound, but similar arguments apply for the lower bound.

$$\begin{aligned}
\Pr[\mathbf{A} \mid \mathbf{B}] &\leq \frac{d(w) + 0.5n^{-c(L)+2}}{d(S) - 0.5n^{-c(L)+2}} \leq \frac{d(w)}{d(S) - 0.5n^{-c(L)+2}} + n^{-c(L)+2} \\
&\leq \frac{d(w)}{d(S) - 0.5d(S)n^{-c(L)+2}} + n^{-c(L)+2} \\
&\leq \frac{d(w)}{d(S)} \cdot \left(1 + \frac{0.5n^{-c(L)+2}}{1 - 0.5n^{-c(L)+2}}\right) + n^{-c(L)+2} \\
&\leq \frac{d(w)}{d(S)} + O(n^{-c(L)+2}).
\end{aligned}$$

Finally, by setting L (e.g., so that $c(L) \geq 5$) to be large enough, and by a union bound over all nodes and all samples, the proof of the theorem is complete. ◀

5.2 Subnetwork Permutation Routing

The problem of permutation routing over subnetworks would be easy if all the nodes of the subnetwork are part of a single (well-)connected component in the network: those nodes could simply run an existing algorithm for permutation routing [31, 32, 14, 12] in that component. In general, however, those nodes could be arbitrarily spread out in the given network, which makes it hard to find efficient routes between the nodes. Towards that end, we consider "simulating" a virtual network $G_v = (S, E_v)$ for the subset of nodes $S \subseteq V$ participating in the routing problem over the given network $G = (V, E)$. In particular, an edge in G_v corresponds to a (not necessarily simple) path in G; at most $O(\log^2 n)$ bits can be sent over an edge in G_v in a virtual round, where one virtual round can be simulated in some bounded number of rounds in G. To further understand the simulation, the notions of "congestion" and "dilation" for a set of paths (in the multicommodity routing literature) are useful.

▶ **Definition 8** (adapted from [32], `Multicommodity-Routing` or `MRoute` problem).

- **Input:** *Network $G = (V, E)$; Source-Terminal Set $\mathcal{D} = \{(s_1, t_1), \dots\}$ where $\forall i \in [|\mathcal{D}|], \{s_i, t_i\} \subseteq V$ and node s_i knows the id of node t_i and a message m_i that needs to be sent to node t_i, where m_i is of size at most $O(\log |V|)$ bits.*
- **Ensure:** *Return a set of paths $\mathcal{P} = \{P_1, \dots, P_{|\mathcal{D}|}\}$ with congestion C and dilation D such that for every i, P_i is a path connecting nodes s_i and t_i, and every node in P_i is aware of its neighbors on P_i.*
- **Notation:** *Consider the following standard notation for this problem.*
  - *Let $C = \max_{e \in E}(\sum_{P \in \mathcal{P}} F(e, P))$ be the* congestion *of the solution, where $F(e, P)$ equates to the number of times, the edge e, appears in the path P.*
  - *Let $D = \max_{P \in \mathcal{P}} |P|$ be the* dilation *of the solution, i.e., the length of the longest path.*
  - *Let $W = \max_{v \in V} |\{sd \mid (v \in sd) \wedge (sd \in \mathcal{D})\}|$ be the* width *of the input.*

The problem of simulating the virtual network $G_v = (S, E_v)$ can be viewed as finding the set of paths in G of the multicommodity routing problem for the set of edges in E_v. Once this set of paths is established with congestion C and dilation D, the elegant randomized-delay technique by Leighton, Maggs and Rao [48] can be used to efficiently send a message from one end of a path to the other; for e.g., in recent years, the following theorem has been useful in the context of low-congestion shortcuts [29, 41].

▶ **Theorem 9** (adapted from [29]). *Given a set of paths $\mathcal{P} = \{P_1, \dots\}$ as the solution for an* `MRoute` *problem with $\mathcal{D} = \{(s_1, t_1), \dots\}$ in a network $G = (V, E)$. Then, for every $i \in [|\mathcal{D}|]$, node s_i can send the message m_i to node t_i over the path P_i in at most $\tilde{O}(C + D)$ rounds.*

Early solutions to permutation routing [31, 32] exploited random walks to construct several layers of such virtual networks on top of each other, forming a hierarchical decomposition of the nodes. Specifically, they require the nodes to embed their ids in tokens, and the tokens execute simple random walks until they are (sufficiently) randomly distributed among the nodes. By doing so, these tokens can be used to form a (low congestion and low dilation) "random graph" virtual network, where the edges are randomly chosen by the nodes.

If the nodes could establish a connection directly to the node id in a token, the virtual network can easily be formed. However, as the network is static, one standard way, for any node to establish a connection to the node id in a token (that executed a random walk and ended at that node), is to *reverse* the token until it reaches the (source) node that started the token (see, e.g., [31, 36]), so that this "token-reversal" can be executed, in parallel, by all nodes. However, depending on the random walk algorithm executed by the tokens, such (reversed) routes may not always have low-congestion.

For instance, if the nodes execute **Amoeba-Walk**$(S, L = \Theta(\tau_{\text{mix}}))$ in network $G = (V, E)$, and the subnetwork size $|S| = \Omega(\sqrt{n})$, and there is some node $u \in S$ with $d(u) = \Omega(\sqrt{n})$ and for all $v \in V \setminus \{u\}$, $d(v) = O(1)$, then the aforementioned token-reversal strategy can cause congestion issues. By design of the virtual network [31, 32], each node w has $\tilde{O}(d(w))$ random virtual connections in the subnetwork (i.e., proportional to its degree). But the path taken by the amoeba-walk token started by node u, before it *first* splits into two tokens, would consist of (at least) one node whose degree is $O(1)$. That node would clearly be a bottleneck when $\tilde{O}(d(u))$ (distinct) tokens of the node u traverse their paths in reverse direction from their destinations. Thus, we would like to avoid such congestion issues in token-reversal, whilst retaining the sampling properties of amoeba-walk (cf. Section 5.1). This is done by splitting the initial count equally in $d(w)$ amoeba-walk tokens at each node $w \in S$; see Algorithm 3.

Algorithm 3 Create-Routes(S, L) for a node u.

```
1:  // Node $u$ simulates $d(u)$ subnodes, indexed from 1 to $d(u)$.
2:  // The id of a subnode is the concatenation of node $u$'s id and its index.
3:  // Note: The notion of subnode ids is merely for analysis exposition.
4:  Let $\alpha = \Theta(1)$ and $\mathcal{C} = \alpha m d(u) \log n / d(S)$ and token $t_i = ((u.id \| i), L, \lceil \mathcal{C}/d(u) \rceil)$;
5:  // Start $d(u)$ amoeba-walks, with *reduced* initial count.
6:  (Round 1) $\wedge$ $(u \in S) \rightarrow \forall i \in [d(u)]$, **LazyRW**$(t_i)$;
7:  **for** each received token $t = (id, l, count)$ **do**
8:      // Mix Phase
9:      $l \geq 1 \rightarrow$ **LazyRW**$((id, l-1, count))$;
10:     **if** $l < 1 \wedge count > 1$ **then**
11:         // Divide Phase
12:         **LazyRW**$((id, L, \lceil count/2 \rceil))$;
13:         **LazyRW**$((id, L, \lfloor count/2 \rfloor))$;
14:     **end if**
15: **end for**
```

▶ **Theorem 4.** *Consider any network $G = (V, E)$ with mixing time τ_{mix} and any set $S \subseteq V$. The* `SPRoute`(S) *problem can be solved in* $\exp(O(\sqrt{\log |S|})) \cdot \tilde{O}(\tau_{\text{mix}})$ *rounds whp.*

Proof. The main idea can be summarized as follows: (1) consider a new "virtual" network, $G_v = (S, E_v)$, with a mixing time of $\tilde{O}(1)$, where one (virtual) round of communication in G_v can be simulated by $\tilde{O}(\tau_{\text{mix}})$ rounds of communication in G, and then, (2) run the best-known distributed algorithm for permutation routing (e.g., [32]) over the virtual network G_v, resulting in an overall round complexity of $\exp(O(\sqrt{\log|S|}))\cdot\tilde{O}(\tau_{\text{mix}})$.

Details about Virtual Network. In the network $G_v = (S, E_v)$, each node u has degree $\Theta(d(u)\log n)$, and at most $O(\log^2 n)$ bits can be exchanged along any edge in any virtual round. The "structure" of G_v (ideally) corresponds to a random $\Theta(\log n)$-out graph over $d(S)$ "subnodes," where each node $u \in S$ simulates $d(u)$ subnodes, and each subnode connects to $\Theta(\log n)$ random subnodes (drawn from a uniform distribution) from the $d(S)$ subnodes. A random k-out graph is obtained when each node connects to k (uniformly) random nodes; for $k = \Theta(\log n)$, such a graph has a mixing time of $\tilde{O}(1)$, with $\tilde{\Omega}(1)$ conductance and $\Theta(\log n)$ max-degree, whp; see, e.g., [27]. Thus, G_v would be an expander[4] graph with a mixing time of $\tilde{O}(1)$ with high probability, where each node $u \in S$ has degree equal to $\Theta(d(u)\log n)$.

Distributing Samples and Creating Routes. Let every node $u \in S$ simulate $d(u)$ subnodes, where each subnode is responsible for sending or receiving a message along an edge of node u, for the `SPRoute`(S) problem. Let the subnodes of any node $u \in S$ be indexed by 1 to $d(u)$, so that the id of a subnode is the concatenation of id of node u and its index. Let S_{sub} be the set of all subnode ids. Let the nodes execute **Create-Routes**$(S, \Theta(\tau_{\text{mix}}))$ algorithm. Finally, let $\mathcal{S}(u)$ be the multi-set of subnode ids (referred to as "samples"), where the multiplicity of a subnode is equal to the total count among all its tokens at node u, at the end of the execution of Algorithm 3.

On Congestion and Subsampling. The congestion and sampling analyses of the amoeba-walk algorithm can be inherited by Algorithm 3. In the first round, each node $u \in V$ receives $O(d(u)\log n)$ distinct tokens (i.e., from different subnodes) whp, due to Line 6 of Algorithm 3. After the first round, the nodes execute the amoeba-walk algorithm, with the only exception that the total number of distinct ids is now equal to the volume of set S, so the congestion analysis (cf. Lemma 7) also holds for Algorithm 3.

Furthermore, since all nodes execute Algorithm 3 for the parameter $L = \Theta(\tau_{\text{mix}})$, the subsampling analysis for amoeba-walk (cf. Theorem 3) holds for the final distribution of the tokens, except that the distribution guarantee would be for the subnodes, instead of nodes. In particular, with high probability, for any node $u \in V$, any sample $s \in \mathcal{S}(u)$, any $w \in S_{sub}$, $\Pr[s = w] = (1/d(S)) \pm n^{-\Omega(1)}$, as the total count started by any node in S is equally divided among its subnodes in round 1 (due to Line 4 of Algorithm 3).

Efficiently Forming the Virtual Network. The nodes in S can use the samples to establish the aforementioned $\Theta(\log n)$-out random graph as the virtual network, which is done by traversing the paths taken by the tokens (that contain those samples) in the reversed direction. To that end, for each round $r \geq 1$ during the execution of Algorithm 3, every node $u \in V$ maintains a dictionary $D_r(\cdot)$: $d(u) \times S_{sub} \to \mathbb{R}$ to "log" the total count (over all tokens) of a certain subnode delivered by a certain neighbor in round r. Such logs are helpful for appropriately choosing a neighbor for sending (back) a message intended to a particular node. Let $ct(t_w) = c$ for any token $t_w = (w \in S_{sub}, L, c)$, and $T_t(w, v, u)$ is a multi-set such that $T_r(w, v, u) = \{t_w \mid t_w$ is sent to node u by node $v \in N(u)$ in round $r\}$. For establishing the token-reversal paths, each node $u \in V$ stores $D_r(v, w) = \sum_{t_w \in T_t(w,v,u)} ct(t_w)$ in round r.

Let $\mathit{Endpoints}(u) \subseteq S$ be the set of nodes that choose node $u \in S$ as their neighbor for the virtual network G_v (via available samples, at the end of Algorithm 3); by design, the size of this set of incoming edges is $|\mathit{Endpoints}(u)| = \Theta(d(u)\log n)$. For each node $u \in S$,

[4] Each node could have $\tilde{O}(1)$ self-loops, but even if the self-loops are discarded, the remaining graph is an expander with $\tilde{\Omega}(1)$ conductance.

and if each node in *Endpoints*(u) sends a (distinct) message, using the path traversed by the sample (i.e., a token containing a subnode of node u), then each of those messages must be forwarded, in the network G (via locally stored dictionaries), to the next node in $\tilde{O}(1)$ rounds (resp.), until all the messages delivered to node u in at most $\tilde{O}(\tau_{\text{mix}})$ rounds.

To see that the token-reversal paths have low-congestion, due to Line 4 of Algorithm 3 (and the sampling guarantees for subnodes), for any node in S, each incoming edge in the virtual network G_v, is (almost-)uniform *randomly mapped* to its set of subnodes, i.e., each *subnode* of a node $u \in S$ is the recipient of $O(\log n)$ distinct (reversal) messages by other nodes in S in any virtual round (or, in other words, $O(\log n)$ incoming virtual edges) with high probability. This key observation implies that in any (real) round, during token-reversal, the number of distinct (reversal) messages at any node $u \in V$ is $O(d(u) \log^2 n)$, with high probability, as in each round of the (forward) execution of Algorithm 3, by Theorem 7, each node $u \in V$ received tokens from $O(d(u) \log n)$ distinct subnodes, with high probability.

Simulation of Virtual Network. Due to the token-forwarding and token-reversal techniques, an edge in G_v is simulated by a path in G of congestion $\tilde{O}(1)$ and dilation $\tilde{O}(\tau_{\text{mix}})$. Thus, by Theorem 9, and using the best-known algorithm for permutation routing [32] over G_v, the `SPRoute`(S) problem can be solved in $\exp(O(\sqrt{\log |S|})) \cdot \tilde{O}(\tau_{\text{mix}})$ rounds over G. ◀

6 Conclusion and Future Work

Motivated by distributed computing for subnetworks, we provide two distributed random walk algorithms and their congestion analyses. As an application, we provide a solution for the subnetwork permutation routing problem that runs in $\exp(O(\sqrt{\log |S|})) \cdot \tilde{O}(\tau_{\text{mix}})$ rounds. Thus, as a consequence, if $|S| = n$, we resort to the state-of-the-art guarantees [32], and for e.g., if $|S| \leq \exp(O(\log^2 \log n))$, our solution runs in $\tilde{O}(\tau_{\text{mix}})$ rounds. In fact, once the routing paths have been established, our solution has an overall message complexity of $\exp(O(\sqrt{\log |S|})) \cdot \tilde{O}(d(S) \cdot \tau_{\text{mix}})$, i.e., the entire network need not even participate in subsequent communication among the nodes in S.

We conclude with a few open questions and directions for future work.

1. By design, the amoeba-walk incurs an (extra) $O(\log n)$ multiplicative factor in the round complexity (i.e., until all tokens have count equal to 1). This is because tokens divide into only *two* tokens in the interval of L rounds. Thus, a natural open question is whether there exists an algorithm that can (randomly) distribute the samples (i.e., node ids) of the subnetwork in at most $\Theta(\tau_{\text{mix}})$ rounds.
2. Our solution for subnetwork permutation routing is randomized, which relies on efficient simulation of a random graph (with good expansion properties) over the subnetwork, and black-box application of an algorithm for permutation routing. Recent works [14, 12] have provided deterministic solutions to permutation routing over expander networks. We believe that it is interesting if one can deterministically simulate an expander graph over the subnetwork, e.g., using a *deterministic analogue* of amoeba-walks.
3. In a breakthrough in packet routing, Haeupler, Räcke and Ghaffari [40] constructed a polylog(n)-competitive *oblivious routing* scheme [62, 63] that minimizes the completion-time (i.e., congestion *and* dilation) of the routes on any graph, whilst giving a distributed *universally-optimal* solution, up to $n^{o(1)}$ factors, for any permutation routing demand, where a node u can be part of $\tilde{\Theta}(d(u))$ sources/terminals. It is interesting if analogous performance guarantees can obtained for any subnetwork premutation demand, where the solution is universally-optimal, up to subpolynomial factors of the *subnetwork size.*

References

1 Alexandr Andoni, Anupam Gupta, and Robert Krauthgamer. Towards $(1 + \epsilon)$-approximate flow sparsifiers. In *Proc. SODA*, pages 279–293, 2014. doi:10.1137/1.9781611973402.20.

2 John Augustine, Mohsen Ghaffari, Robert Gmyr, Kristian Hinnenthal, Christian Scheideler, Fabian Kuhn, and Jason Li. Distributed computation in node-capacitated networks. In *Proc. SPAA*, pages 69–79, 2019. doi:10.1145/3323165.3323195.

3 John Augustine, Anisur Rahaman Molla, Ehab Morsy, Gopal Pandurangan, Peter Robinson, and Eli Upfal. Storage and search in dynamic peer-to-peer networks. In *Proc. SPAA*, pages 53–62, 2013. doi:10.1145/2486159.2486170.

4 John Augustine, Gopal Pandurangan, and Peter Robinson. Fast byzantine agreement in dynamic networks. In *Proc. PODC*, pages 74–83, 2013. doi:10.1145/2484239.2484275.

5 John Augustine, Gopal Pandurangan, and Peter Robinson. Fast byzantine leader election in dynamic networks. In *Proc. DISC*, pages 276–291, 2015. doi:10.1007/978-3-662-48653-5_19.

6 John Augustine, Gopal Pandurangan, Peter Robinson, Scott T. Roche, and Eli Upfal. Enabling robust and efficient distributed computation in dynamic peer-to-peer networks. In *Proc. FOCS*, pages 350–369, 2015. doi:10.1109/FOCS.2015.29.

7 John Augustine and Sumathi Sivasubramaniam. Spartan: A framework for sparse robust addressable networks. In *Proc. IPDPS*, pages 1060–1069, 2018. doi:10.1109/IPDPS.2018.00115.

8 Tugkan Batu, Amitabh Trehan, and Chhaya Trehan. All you need are random walks: Fast and simple distributed conductance testing. In *Proc. SIROCCO*, pages 64–82, 2024. doi:10.1007/978-3-031-60603-8_4.

9 Petra Berenbrink, George Giakkoupis, and Peter Kling. Tight bounds for coalescing-branching random walks on regular graphs. In *Proc. SODA*, pages 1715–1733, 2018. doi:10.1137/1.9781611975031.112.

10 Keren Censor-Hillel, Eldar Fischer, Gregory Schwartzman, and Yadu Vasudev. Fast distributed algorithms for testing graph properties. *Distributed Comput.*, 32(1):41–57, 2019. doi:10.1007/S00446-018-0324-8.

11 Keren Censor-Hillel, Dean Leitersdorf, and Volodymyr Polosukhin. On sparsity awareness in distributed computations. In *Proc. SPAA*, pages 151–161, 2021. doi:10.1145/3409964.3461798.

12 Yi-Jun Chang, Shang-En Huang, and Hsin-Hao Su. Deterministic expander routing: Faster and more versatile. In *Proc. PODC*, pages 194–204, 2024. doi:10.1145/3662158.3662797.

13 Yi-Jun Chang, Seth Pettie, Thatchaphol Saranurak, and Hengjie Zhang. Near-optimal distributed triangle enumeration via expander decompositions. *J. ACM*, 68(3):21:1–21:36, 2021. doi:10.1145/3446330.

14 Yi-Jun Chang and Thatchaphol Saranurak. Deterministic distributed expander decomposition and routing with applications in distributed derandomization. In *Proc. FOCS*, pages 377–388, 2020. doi:10.1109/FOCS46700.2020.00043.

15 Moses Charikar, Tom Leighton, Shi Li, and Ankur Moitra. Vertex sparsifiers and abstract rounding algorithms. In *Proc. FOCS*, pages 265–274, 2010. doi:10.1109/FOCS.2010.32.

16 Soumyottam Chatterjee, Gopal Pandurangan, and Nguyen Dinh Pham. Distributed MST: A smoothed analysis. In *Proc. ICDCN*, pages 15:1–15:10, 2020. doi:10.1145/3369740.3369778.

17 Yu Chen and Zihan Tan. On $(1 + \epsilon)$-approximate flow sparsifiers. In *Proc. SODA*, pages 279–293, 2024. doi:10.1137/1.9781611977912.63.

18 Julia Chuzhoy. On vertex sparsifiers with steiner nodes. In *Proc. STOC*, pages 673–688, 2012. doi:10.1145/2213977.2214039.

19 Colin Cooper, Tomasz Radzik, and Nicolas Rivera. The coalescing-branching random walk on expanders and the dual epidemic process. In *Proc. PODC*, pages 461–467, 2016. doi:10.1145/2933057.2933119.

20 Colin Cooper, Tomasz Radzik, and Nicolas Rivera. Improved cover time bounds for the coalescing-branching random walk on graphs. In *Proc. SPAA*, pages 305–312, 2017. doi:10.1145/3087556.3087564.

21 Maximilian Drees, Robert Gmyr, and Christian Scheideler. Churn- and dos-resistant overlay networks based on network reconfiguration. In *Proc. SPAA*, pages 417–427, 2016. doi:10.1145/2935764.2935783.

22 Chinmoy Dutta, Gopal Pandurangan, Rajmohan Rajaraman, and Scott T. Roche. Coalescing-branching random walks on graphs. In *Proc. SPAA*, pages 176–185, 2013. doi:10.1145/2486159.2486197.

23 Chinmoy Dutta, Gopal Pandurangan, Rajmohan Rajaraman, and Scott T. Roche. Coalescing-branching random walks on graphs. *ACM Trans. Parallel Comput.*, 2(3):20:1–20:29, 2015. doi:10.1145/2817830.

24 Matthias Englert, Anupam Gupta, Robert Krauthgamer, Harald Räcke, Inbal Talgam-Cohen, and Kunal Talwar. Vertex sparsifiers: New results from old techniques. In *Proc. APPROX-RANDOM*, pages 152–165, 2010. doi:10.1007/978-3-642-15369-3_12.

25 Laurent Feuilloley, Juho Hirvonen, and Jukka Suomela. Locally optimal load balancing. In *Proc. DISC*, pages 544–558, 2015. doi:10.1007/978-3-662-48653-5_36.

26 Hendrik Fichtenberger and Yadu Vasudev. A two-sided error distributed property tester for conductance. In *Proc. MFCS*, pages 19:1–19:15, 2018. doi:10.4230/LIPICS.MFCS.2018.19.

27 Abraham D. Flaxman. Expansion and lack thereof in randomly perturbed graphs. *Internet Math.*, 4(2):131–147, 2007. doi:10.1080/15427951.2007.10129290.

28 Mohsen Ghaffari. An improved distributed algorithm for maximal independent set. In *Proc. SODA*, pages 270–277, 2016. doi:10.1137/1.9781611974331.CH20.

29 Mohsen Ghaffari and Bernhard Haeupler. Distributed algorithms for planar networks II: low-congestion shortcuts, mst, and min-cut. In *Proc. SODA*, pages 202–219, 2016. doi:10.1137/1.9781611974331.CH16.

30 Mohsen Ghaffari, Bernhard Haeupler, and Goran Zuzic. Hop-constrained oblivious routing. In *Proc. STOC*, pages 1208–1220, 2021. doi:10.1145/3406325.3451098.

31 Mohsen Ghaffari, Fabian Kuhn, and Hsin-Hao Su. Distributed MST and routing in almost mixing time. In *Proc. PODC*, pages 131–140, 2017. doi:10.1145/3087801.3087827.

32 Mohsen Ghaffari and Jason Li. New distributed algorithms in almost mixing time via transformations from parallel algorithms. In *Proc. DISC*, pages 31:1–31:16, 2018. doi:10.4230/LIPICS.DISC.2018.31.

33 George Giakkoupis, Frederik Mallmann-Trenn, and Hayk Saribekyan. How to spread a rumor: Call your neighbors or take a walk? In *Proc. PODC*, pages 24–33, 2019. doi:10.1145/3293611.3331622.

34 George Giakkoupis, Hayk Saribekyan, and Thomas Sauerwald. Spread of information and diseases via random walks in sparse graphs. In *Proc. DISC*, pages 9:1–9:17, 2020. doi:10.4230/LIPICS.DISC.2020.9.

35 Seth Gilbert, Gopal Pandurangan, Peter Robinson, and Amitabh Trehan. Dconstructor: Efficient and robust network construction with polylogarithmic overhead. In *Proc. PODC*, pages 438–447, 2020. doi:10.1145/3382734.3405716.

36 Seth Gilbert, Peter Robinson, and Suman Sourav. Leader election in well-connected graphs. *Algorithmica*, 85(4):1029–1066, 2023. doi:10.1007/S00453-022-01068-X.

37 Rachid Guerraoui, Florian Huc, and Anne-Marie Kermarrec. Highly dynamic distributed computing with byzantine failures. In *Proc. PODC*, pages 176–183, 2013. doi:10.1145/2484239.2484263.

38 Bernhard Haeupler and Dahlia Malkhi. Distributed resource discovery in sub-logarithmic time. In *Proc. PODC*, pages 413–419, 2015. doi:10.1145/2767386.2767435.

39 Bernhard Haeupler, Gopal Pandurangan, David Peleg, Rajmohan Rajaraman, and Zhifeng Sun. Discovery through gossip. In *Proc. SPAA*, pages 140–149, 2012. doi:10.1145/2312005.2312031.

40 Bernhard Haeupler, Harald Räcke, and Mohsen Ghaffari. Hop-constrained expander decompositions, oblivious routing, and distributed universal optimality. In *Proc. STOC*, pages 1325–1338, 2022. doi:10.1145/3519935.3520026.

41 Bernhard Haeupler, David Wajc, and Goran Zuzic. Universally-optimal distributed algorithms for known topologies. In *Proc. STOC*, pages 1166–1179, 2021. doi:10.1145/3406325.3451081.

42 Stephan Holzer and Roger Wattenhofer. Optimal distributed all pairs shortest paths and applications. In *Proc. PODC*, pages 355–364, 2012. doi:10.1145/2332432.2332504.

43 Anna R. Karlin and Eli Upfal. Parallel hashing: an efficient implementation of shared memory. *J. ACM*, 35(4):876–892, 1988. doi:10.1145/48014.350550.

44 Robert Krauthgamer and Ron Mosenzon. Exact flow sparsification requires unbounded size. In *Proc. SODA*, pages 2354–2367, 2023. doi:10.1137/1.9781611977554.CH91.

45 Shay Kutten, Gopal Pandurangan, David Peleg, Peter Robinson, and Amitabh Trehan. Sublinear bounds for randomized leader election. *Theor. Comput. Sci.*, 561:134–143, 2015. doi:10.1016/J.TCS.2014.02.009.

46 Jakub Lacki, Slobodan Mitrovic, Krzysztof Onak, and Piotr Sankowski. Walking randomly, massively, and efficiently. In *Proc. STOC*, pages 364–377, 2020. doi:10.1145/3357713.3384303.

47 Ching Law and Kai-Yeung Siu. Distributed construction of random expander networks. In *Proc. INFOCOM*, pages 2133–2143, 2003. doi:10.1109/INFCOM.2003.1209234.

48 Frank Thomson Leighton, Bruce M. Maggs, and Satish Rao. Universal packet routing algorithms (extended abstract). In *Proc. FOCS*, pages 256–269, 1988. doi:10.1109/SFCS.1988.21942.

49 Frank Thomson Leighton and Ankur Moitra. Extensions and limits to vertex sparsification. In *Proc. STOC*, pages 47–56, 2010. doi:10.1145/1806689.1806698.

50 Christoph Lenzen and Boaz Patt-Shamir. Improved distributed steiner forest construction. In *Proc. PODC*, pages 262–271, 2014. doi:10.1145/2611462.2611464.

51 David A. Levin, Yuval Peres, and Elizabeth L. Wilmer. *Markov Chains and Mixing Times*. Amer. Math. Soc., 2nd edition, 2017. URL: https://pages.uoregon.edu/dlevin/MARKOV/.

52 Konstantin Makarychev and Yury Makarychev. Metric extension operators, vertex sparsifiers and lipschitz extendability. In *Proc. FOCS*, pages 255–264, 2010. doi:10.1109/FOCS.2010.31.

53 Laurent Massoulié, Erwan Le Merrer, Anne-Marie Kermarrec, and Ayalvadi J. Ganesh. Peer counting and sampling in overlay networks: random walk methods. In *Proc. PODC*, pages 123–132, 2006. doi:10.1145/1146381.1146402.

54 Michael Mitzenmacher, Rajmohan Rajaraman, and Scott T. Roche. Better bounds for coalescing-branching random walks. In *Proc. SPAA*, pages 313–323, 2016. doi:10.1145/2935764.2935791.

55 Michael Mitzenmacher, Rajmohan Rajaraman, and Scott T. Roche. Better bounds for coalescing-branching random walks. *ACM Trans. Parallel Comput.*, 5(1):2:1–2:23, 2018. doi:10.1145/3209688.

56 Michael Mitzenmacher and Eli Upfal. *Probability and Computing: Randomized Algorithms and Probabilistic Analysis*. Cambridge University Press, 2005. doi:10.1017/CBO9780511813603.

57 Ankur Moitra. Approximation algorithms for multicommodity-type problems with guarantees independent of the graph size. In *Proc. FOCS*, pages 3–12, 2009. doi:10.1109/FOCS.2009.28.

58 Ankur Moitra. *Vertex sparsification and universal rounding algorithms*. PhD thesis, Massachusetts Institute of Technology, Cambridge, MA, USA, 2011. URL: https://hdl.handle.net/1721.1/66019.

59 Anisur Rahaman Molla and Gopal Pandurangan. Distributed computation of mixing time. In *Proc. ICDCN*, page 5, 2017. URL: http://dl.acm.org/citation.cfm?id=3007784.

60 Gopal Pandurangan, Peter Robinson, and Michele Scquizzato. A time- and message-optimal distributed algorithm for minimum spanning trees. In *Proc. STOC*, pages 743–756, 2017. doi:10.1145/3055399.3055449.

61 David Peleg. *Distributed computing: a locality-sensitive approach.* SIAM, 2000. doi:10.1137/1.9780898719772.

62 Harald Räcke. Minimizing congestion in general networks. In *Proc. FOCS*, pages 43–52, 2002. doi:10.1109/SFCS.2002.1181881.

63 Harald Räcke. Optimal hierarchical decompositions for congestion minimization in networks. In *Proc. STOC*, pages 255–264, 2008. doi:10.1145/1374376.1374415.

64 Abhiram G. Ranade. How to emulate shared memory. *J. Comput. Syst. Sci.*, 42(3):307–326, 1991. doi:10.1016/0022-0000(91)90005-P.

65 Parikshit Saikia and Sushanta Karmakar. Improved distributed approximation for steiner tree in the CONGEST model. *J. Parallel Distributed Comput.*, 158:196–212, 2021. doi:10.1016/J.JPDC.2021.08.004.

66 Atish Das Sarma, Anisur Rahaman Molla, and Gopal Pandurangan. Fast distributed computation in dynamic networks via random walks. In *Proc. DISC*, pages 136–150, 2012. doi:10.1007/978-3-642-33651-5_10.

67 Hsin-Hao Su and Hoa T. Vu. Distributed data summarization in well-connected networks. In *Proc. DISC*, pages 33:1–33:16, 2019. doi:10.4230/LIPICS.DISC.2019.33.

68 Leslie G. Valiant and Gordon J. Brebner. Universal schemes for parallel communication. In *Proc. STOC*, pages 263–277, 1981. doi:10.1145/800076.802479.

A Tools in Probability Theory

We exploit the following Chernoff bounds (see, Mitzenmacher and Upfal [56]) in our paper.

▶ **Theorem 10.** *Let* $X = \sum_1^n X_i$ *and* $\mu = \mathbb{E}[X]$ *where* $X_1, \ldots, X_n$ *are binary independent random variables. Then for any* $0 < \delta \leq 1, \Pr(|X - \mu| \geq \delta\mu) \leq 2e^{-\mu\delta^2/3}$.

Moreover, we recall the definition of conditional probability [56].

▶ **Definition 11.** *The conditional probability that event* $\mathbf{E}$ *occurs given that event* $\mathbf{F}$ *occurs is defined as,* $\Pr[\mathbf{E} \mid \mathbf{F}] = \Pr[\mathbf{E} \cap \mathbf{F}]/\Pr[\mathbf{F}]$, *where* $\Pr[\mathbf{F}] \neq 0$.

B Congestion Analysis of Count Random Walks

In this section, we provide a congestion analysis for the count random walk algorithm (see, Kutten et al. [45]) that can be started by any subset of nodes $S \subseteq V$ in a network $G = (V, E)$.

Similar to Theorem 5, the key observation is that if the subnetwork size $|S|$ is large, and if the initial count in each token is also large (despite the total count being not more than $\Theta(m \log n)$), then some node may have to send $\Omega(\log^c n)$ bits over an edge in some round (for any constant c) with high probability, violating the congestion constraints.

▶ **Theorem 12.** *Consider a network* $G = (V, E)$ *with conductance* $\Phi = \Omega(1)$ *and max-degree* $\Delta = O(1)$*. There is a subset of nodes* $S \subset V$ *in which at least one node sends* $\Omega(\log^c n)$ *bits with high probability, for any constant c, along an edge in some round during the execution of* **CountRW**(S) *algorithm.*

Proof. Similar to the proof of Theorem 5, the key idea is to shift our focus from a collection of tokens to individual tokens of count value equal to one from round 1, where *each* count is *independently* executing a lazy random walk.

Algorithm 4 CountRW(S) for a node u.

```
1: Let $N(u,i)$ denote $w \in N(u)$ such that $|\{w' \mid w' < w \wedge w' \in N(u)\}| = i - 1$;
2: Let $\alpha = \Theta(1)$ and $\mathcal{C} = \lceil \alpha m d(u) \log n / d(S) \rceil$ and token $t = (u.id, \mathcal{C})$;
3: (Round 1) $\wedge$ ($u \in S$) $\rightarrow$ **LazyRW**($t$);
4: **for** set of all tokens $T_{id} = \{(id, c_1), (id, c_2), \dots\}$ of node $id$ received in this round **do**
5:     $count := \sum_{(id,l,c) \in T_{id}} c$; // Aggregate the counts from all the tokens of node $id$
6:     Let $bins := [0, \dots, 0]$ where size($bins$) $= d(u) + 1$;
7:     **while** $count \geq 1$ **do**
8:         $rand \in_R [d(u)]$ and $b \in_R \{0,1\}$ are drawn randomly from $[d(u)]$ and $\{0,1\}$ resp.;
9:         $b = 0 \rightarrow bins[rand] := bins[rand] + 1$;
10:        $b = 1 \rightarrow bins[0] := bins[0] + 1$;
11:        $count := count - 1$;
12:    **end while**
13:    // Each "count" in the token executes a lazy random walk via the above process
14:    $bins[0] > 0 \rightarrow$ Send token $t = (id, bins[0])$ to self;
15:    $\forall i \in [d(u)], bins[i] > 0 \rightarrow$ Send token $t = (id, bins[i])$ to node $N(u,i)$;
16: **end for**
```

Notation. Let the initial count (in round 1), started by nodes in S, be denoted as $\mathcal{C} \in \mathbb{N}$. For the sake of analysis, the initial count started by node in S, is viewed as a *collection* of individual counts, indexed by $[\mathcal{C}]$. Let $C_t(u,v)$ be the random variable for the total count of node u (over all tokens present) at any node $v \in V$ in any round $t \geq 1$. Let $C_t(u,v,k)$ be the indicator random variables for the presence of the count k of node $u \in S$ at any node $v \in V$ in any round $t \geq 1$. Let $\mathrm{P}_{i,j}(u,v)$ denote the probability that a lazy random walk that starts at node u in round i and ends at node v in round j, for $j > i$. Let $\mathrm{sgn}(x) = 0$ if x is even, and 1 otherwise. Let $B(u,r) = \{v \mid 0 \leq dist(u,v) \leq r\}$ be the ball of radius r around a node u, where $dist(u,v)$ denotes the distance between nodes u and v in G. Let $\epsilon = \Phi/\Delta$ and $h = \lceil \log_{1+\epsilon} 10\mathcal{B} \rceil$ where $\mathcal{B} = \Omega(\log^c n)$; note $\forall u \in V, |B(u,H)| \geq 10\mathcal{B}$.

Setting up the Bottleneck. Consider some node $v_0 \in V$ and $|S| = \lfloor \sqrt{n} \rfloor$. Let $r_0 = \mathrm{argmin}_r(|B(v_0,r)| \geq |S|)$. Let S be the set of nodes *closest* to the node v_0 among nodes in $B(v_0,r_0)$, including node v_0, where $|S| \geq \lfloor \sqrt{n} \rfloor$. Let $S_{even} = \{w \mid \mathrm{sgn}(dist(v_0,w)) = 0 \wedge (w \in S)\}$ and $S_{odd} = S \setminus S_{even}$. By pigeonhole principle, either $S_{even} \geq 5\mathcal{B}$ or $S_{odd} \geq 5\mathcal{B}$. If $S_{even} \geq 5\mathcal{B}$, then $S' = S_{even}$ and $b = 0$, whereas if $S_{odd} \geq 5\mathcal{B}$, then $S' = S_{odd}$ and $b = 1$. Furthermore, if $\mathrm{sgn}(h) = b$, then $h' = h$, whereas if $\mathrm{sgn}(h) \neq b$, then $h' = h - 1$.

Analysis. As each count is executing a lazy random walk, for any nodes $u \in S$ and $v \in V$, and any count $k \in [\mathcal{C}]$, $\mathbb{E}[C_t(u,v,k)] = \mathrm{P}_{1,t}(u,v) \geq 1/(2\Delta)^t$. By linearity of expectation, and due to max-degree Δ, the following two claims hold: (1) $\forall u \in S$, if $0 < dist(u,v_0) = t$, then $\mathbb{E}[C_t(u,v_0)] \geq \lfloor \mathcal{C}/(2\Delta)^t \rfloor$, and (2) $\forall u \in S$, if $0 \leq dist(u,v_0) \leq t \leq h-2$ and $\mathrm{sgn}(t) = dist(u,v_0)$, then $\mathbb{E}[C_{t+2}(u,v_0)] \geq \mathbb{E}[C_t(u,v)]/4\Delta^2$. By combining the two claims, and the bound on conductance, $\forall u \in S'$, $\mathbb{E}[C_{h'}(u,v_0)] \geq \lfloor \Theta(\sqrt{n}\log n)/(2\Delta)^{h'} \rfloor = \Omega(\log n)$, as $\Delta = O(1)$ and $\Phi = \Omega(1)$, $(2\Delta)^{\log_{1+\epsilon} 10\mathcal{B}} \ll \sqrt{n}$. Thus, the node v_0 needs to send tokens from $\mathcal{B} = \Omega(\log^c n)$ distinct nodes in round $h'+1$, in expectation.

Moreover, as $C_t(u,v)$ is defined as the sum of counts from node u to node v in round t, it can be viewed as a sum of (independent) random walks that started at node u and ended at node v in round t. To that end, $C_t(u,v_0) = \sum_{i \in [\mathcal{C}]} C_t(u,v_0,i)$ is the sum of independent binary random variables; by Chernoff bounds, $C_{h'}(u,v_0) = \Omega(\log n)$ whp. By a union bound, this holds true for any pair u, v_0 where $u \in S'$. Thus, for any constant c, the node v_0 needs to send tokens from $\mathcal{B} = \Omega(\log^c n)$ distinct nodes in round $h'+1$, with high probability. ◀