Distributed Construction of Random Expander Networks

Ching Law
Massachusetts Institute of Technology
Cambridge, MA 02139
ching@mit.edu

Kai-Yeung Siu
Massachusetts Institute of Technology
Cambridge, MA 02139
siu@sunny.mit.edu

***Abstract*— We present a novel distributed algorithm for constructing random overlay networks that are composed of d Hamilton cycles. The protocol is completely decentralized as no globally-known server is required. The constructed topologies are expanders with $O(\log_d n)$ diameter with high probability.**

Our construction is highly scalable because both the processing and the space requirements at each node grow logarithmically with the network size. A new node can join the network in $O(\log_d n)$ time with $O(d \log_d n)$ messages. A node can leave in $O(1)$ time with $O(d)$ messages. The protocol is robust against an offline adversary selecting the sequence of the join and leave operations.

We also discuss a layered construction of the random expander networks in which any node can be located in $O(\log n)$ time.

The random expander networks have applications in community discovery, distributed lookup service, and dynamic connectivity.

I. Introduction

The area of peer-to-peer networking has recently gained much attention in both the industry and the research community. Well-known peer-to-peer networks include Napster, Gnutella, Freenet, FastTrack, and eDonkey. However, most of these systems either require a centralized directory or cannot scale beyond a moderate number of nodes. The problem of finding an efficient distributed scalable solution has attracted a lot of research interests [1], [2], [3], [4]. This paper introduces a distributed algorithm for constructing expander networks, which are suitable for peer-to-peer networking, without using any globally-known server.

Interesting properties and algorithms have been discovered for random regular graphs [5], [6], [7], [8]. In particular, it has been found that random regular graphs are expected to have big eigenvalue gaps [9] with high probability, and thus are good expanders.

In this paper, we form expander graphs by constructing a class of regular graphs which we call $\mathbb{H}$-graphs. An $\mathbb{H}$-graph is a $2d$-regular multigraph in which the set of edges is composed of d Hamilton cycles (Figure 1 is an example). Using random walk as a sampling algorithm, a node can join an $\mathbb{H}$-graph in $O(\log_d n)$ time with $O(d \log_d n)$ messages, and leave in $O(1)$ time with $O(d)$ messages.

If we want to search the nodes by their identifiers, we can overlap multiple layers of $\mathbb{H}$-graphs. On layered $\mathbb{H}$-graphs with uniformly distributed identifiers, a join operation takes $O(\log n)$ time and $O(\log n)$ messages, while a leave operation takes $O(1)$ time and $O(\log n)$ messages.

This work is supported in part by the Auto-ID Center.

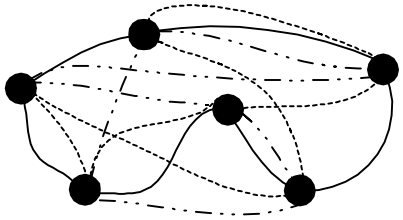

Fig. 1
An $\mathbb{H}$-graph consisting of 3 Hamilton cycles.

Section II describes our network model and gives an overview of the design of $\mathbb{H}$-graphs. Section III introduces the protocol for constructing $\mathbb{H}$-graphs. We discuss several perfect sampling algorithms in Section IV and a random-walk sampling algorithm in Section V, with simulation results in Section VI. Two useful maintenance algorithms are discussed in Section VII. We present layered $\mathbb{H}$-graphs in Section VIII. We discuss applications in Section IX and related work in Section X, and conclude with remarks on future work in Section XI. In this conference paper, most of the proofs are omitted for brevity.

II. Preliminaries

In this section, we will state our assumptions of the underlying network model and then describe the goals and constraints that lead to our design based on random regular graphs.

A. Network Model

We assume a network environment where any node u can send a message to node v as long as node u knows the address of v. If a node fails to receive a message for whatever reason, the sending node can repeat sending the message without causing the algorithm to fail. There can be node failures but not permanent link failures. Such model is assumed in [10] and most peer-to-peer research. An example of such an underlying network is the Internet when using some reliable messaging protocol such as TCP.

The space requirement will be expressed in the number of addresses. Theoretically, $O(\log N)$ bits are required to encode the address of a node, where N is the largest possible number of nodes. However, for all practical purposes, we can assume that the length of an address is effectively constant (e.g., 128 bits in IPv6).

We assume that there is a maximum delay between all pairs of nodes in the underlying network. In practice, the processing time per message is usually insignificant compared to the communication time. Also, for a small message, the delivery time is mostly independent of the message size.

Each execution of a distributed algorithm will lead to a set of messages sent among the nodes in the graph. The message complexity of an algorithm is the size of this set. Some of these messages have causal relationships: there is some sequence $m_1, m_2, \ldots$ of messages where m_i cannot be sent before m_{i-1} has been received. We will express the time complexity of an algorithm as the length of the longest such sequence.

We will be concerned with the logical topologies overlayed on top of the underlying network. On a set of nodes with labels $[n] = \{1, 2, \ldots, n\}$, a topology can be effectively determined by sets of *neighbors* $N(u)$, which are nodes known to node u (not including u itself), for $u \in [n]$. In the rest of this paper, we represent such logical topology as a graph $G = (V, E)$, where $V = [n]$ and $(u, v) \in E$ if and only if $v \in N(u)$.

We consider distributed algorithms on the graph such that u can send a message to v only if $(u, v) \in E$. During the execution of our algorithm, an edge (u, v) can be added into E if u is informed of the address of v.

B. Goals and Requirements

Our first goal is to construct logical topologies that can support broadcasting and searching efficiently. Our second goal is to make our construction highly scalable. This implies the objectives:

A-1 Resources consumed at each individual node should be as low as possible.

A-2 Time complexities for joining and leaving of nodes should be as low as possible.

A key factor on the load of a node is the number of nodes that it has to communicate with. This parameter determines a node's minimum storage requirement and the maximum number of potential simultaneous network connections. The number of neighbors of a node is its degree in the graph. Therefore, objective A-1 dictates that the degrees should be as small as possible. In order to achieve objective A-2, we need to make sure that the algorithms for nodes joining and leaving are efficient.

C. A Random Graph Approach

To make worst-case scenarios unlikely, we have decided to construct the graph with a randomized protocol. We would like our topology to be 'symmetric' in the sense that every node shares an equal amount of responsibilities. For this reason and for providing better fault tolerance, we did not choose a hierarchical approach. After considering various random graph models, we chose regular multi-graphs that are composed of independent Hamilton cycles. Regular graphs are chosen because we would like the degree to be bounded. Also, random walking on regular graphs, which is a key part of our protocol, has a uniform stationary distribution. Graphs composed of Hamilton cycles have the advantage that the join and leave operations only require local changes in the graph.

Now we shall define the set of $\mathbb{H}$-graphs. Let $\mathbb{H}_n$ denote the set of all Hamilton cycles on set $[n]$. We shall assume $n \geq 3$ for rest of this paper. Consider a multigraph $G = (V, E)$, such that $V = [n]$ and $E = (C_1, \ldots, C_d)$, where $C_1, C_2, \ldots, C_d \in \mathbb{H}_n$. Let $\mathbb{H}_{n,2d}$ be the set of all such $2d$-regular multigraphs. We call the elements in $\mathbb{H}_{n,2d}$ the $\mathbb{H}$-graphs. It can be derived that $|\mathbb{H}_n| = (n-1)!/2$ and $|\mathbb{H}_{n,2d}| = ((n-1)!/2)^d$. If $C_1, C_2, \ldots, C_d$ are independent uniform samples of $\mathbb{H}_n$, then $(V, (C_1, \ldots, C_d))$ is a uniform sample of $\mathbb{H}_{n,2d}$.

Following the notation in Bollobás [11], a probability space is a triple (Ω, Σ, P), where Ω is a finite set, Σ is the set of all subsets of Ω, and P is a measure on Σ such that $P(\Omega) = 1$ and $P(A) = \sum_{w \in A} P(\{w\})$ for any $A \in \Sigma$. In other words, P is determined by the values of $P(\{w\})$ for $w \in \Omega$. For simplicity, we will write $P(w)$ for $P(\{w\})$.

Let U_Ω be the uniform measure on set Ω so that $U_\Omega\{w\} = 1/|\Omega|$ for all $w \in \Omega$. For example, we have $U_{\mathbb{H}_{n,2d}}\{G\} = (2/(n-1)!)^d$ for all $G \in \mathbb{H}_{n,2d}$.

We shall consider two basic operations for a randomized topology construction protocol: JOIN and LEAVE. A JOIN(u) operation inserts node u into the graph. Any node in G should be able to accept a JOIN request at any time. Any node in G can also call LEAVE to remove itself from the graph G. Our algorithms of JOIN and LEAVE are described in Section III. Given an initial probability space $\mathcal{S}_0$ and a sequence of JOIN and LEAVE requests, a randomized topology construction protocol will create a sequence of spaces $\mathcal{S}_1, \mathcal{S}_2, \ldots$.

Friedman [9], [12] showed that a graph chosen uniformly from $\mathbb{H}_{n,2d}$ is very unlikely to have a large second largest eigenvalue. In order to apply Friedman's theorem, we need a protocol that would produce a sequence of uniformly distributed spaces.

Given a probability space $\mathcal{S} = (\Omega', \Sigma', P')$, let $\Omega[\mathcal{S}] = \Omega'$ and $P[\mathcal{S}] = P'$. A probability space $\mathcal{S}$ is uniformly distributed if $P[\mathcal{S}] = U_{\Omega[\mathcal{S}]}$. We would like to have a protocol that creates a sequence of uniformly distributed probability spaces, given *any* sequence of JOIN and LEAVE requests. In addition, a new node should be free to call JOIN at any existing node in the graph.

Summarizing the objectives we have so far, we would like to have a protocol where

B-1 Low space complexity at any node.

B-2 Low time complexities for JOIN and LEAVE.

B-3 Low message complexities for JOIN and LEAVE.

B-4 The probability spaces produced are uniformly distributed.

Satisfying the first three properties is crucial because they are necessary for our protocol to be highly scalable. When property B-4 is not satisfied, we can try to construct sequences

of probability spaces $\mathcal{S}_0, \mathcal{S}_1, \ldots$ having distributions close to be uniform. This can be achieved by an algorithm based on random walks presented in Section V. We have yet to find a protocol satisfying all four properties simultaneously.

III. Construction

In this section we introduce a framework for constructing a random regular network.

The graphs that we shall construct are $2d$-regular multi-graphs in $\mathbb{H}_{n,2d}$, for $d \geq 4$. The neighbors of a node are labeled as $\mathsf{nbr}_{-1}, \mathsf{nbr}_1, \mathsf{nbr}_{-2}, \ldots, \mathsf{nbr}_{-d}, \mathsf{nbr}_d$. For each i, nbr_{-i} and nbr_i denote a node's predecessor and successor on the ith Hamilton cycle (which will be referred to as the level-i cycle).

We start with 3 nodes, because there is only one possible $\mathbb{H}$-graph of size 3. In practice, we might want to use a different topology such as a complete graph when the graph is small.

The graph grows incrementally when new nodes call Join at existing nodes. Any node can leave the graph by calling Leave.

In the following algorithmic pseudocodes, the variable self identifies the node executing the procedure. All the actions are performed by self by default. The expression $u \Rightarrow$Proc() invokes a remote procedure call Proc at node u. The call is assumed to be non-blocking unless a return value is expected. Expression $u \Rightarrow \mathsf{var}$ means that we request the value var from node u. Expression $(u \Rightarrow \mathsf{var}) \leftarrow x$ means that we set the variable var of node u to value x. Thus, messages are exchanged between node self and node u.

Procedure Link connects two nodes on the level-i cycle.

```
Link(u, v, i)
1  (u ⇒ nbr_i) ← v
2  (v ⇒ nbr_{-i}) ← u
```

Procedure Insert(u, i) makes node u the successor of self on the level-i cycle. We assume that Insert is atomic. This can achieved by having a lock for each of the d Hamilton cycles at each node.

```
Insert(u, i)
1  Link(u, self ⇒ nbr_i, i)
2  Link(self, u, i)
```

A new node u joins by calling Join(u) at any existing node in the graph. Node u will be inserted into cycle i between node v_i and node $(v_i \Rightarrow \mathsf{nbr}_i)$ for randomly chosen v_i's for $i = 1, \ldots, d$.

```
Join(u)
1  for i ← 1, ..., d in parallel
2  do v_i ← Sample()
3  for i ← 1, ..., d in parallel
4  do v_i ⇒ Insert(u, i)
```

Procedure Sample() returns a node of the graph chosen uniformly at random. Implementations of Sample are presented in Section IV.

```
Leave()
1  for i ← 1, ..., d in parallel
2  do Link(nbr_{-i}, nbr_i, i)
```

Theorem 1 shows that procedures Join and Leave can preserve uniform probability spaces.

Theorem 1: Let $\mathcal{S}_0, \mathcal{S}_1, \mathcal{S}_2, \ldots$ be a sequence of probability spaces such that

- $\mathcal{S}_0$ is uniformly distributed and $\Omega[\mathcal{S}_0] = \mathbb{H}_{k,2d}$ for some k;
- $\mathcal{S}_{i+1}$ is formed from $\mathcal{S}_i$ by Join or Leave;
- $|\Omega[\mathcal{S}_i]| \geq 3$ for all $i \geq 0$.

Then $\mathcal{S}_i$ is uniformly distributed for all $i \geq 0$.

A graph G of size n has a corresponding n by n matrix A, in which the entry A_{ij} is the number of edges from node i to node j. Let $\lambda(G)$ be the second largest eigenvalue of graph G's matrix. Friedman [9] showed that random regular graphs have close to optimal $\lambda(G)$ with high probability. Although he mainly considered the graphs that are composed of d random permutations, he also showed that his results hold for graphs composed of d random Hamilton cycles. Theorem 2 restates a recent improvement [12] over [9].

Theorem 2 (Friedman): Let G be a graph chosen from $\mathbb{H}_{n,2d}$ uniformly at random. For any $\epsilon > 0$, $\lambda(G) \leq 2\sqrt{2d-1} + \epsilon$ with probability $1 - O(n^{-p})$, where p depends on d.

It has been known that $\lambda(G) \geq 2\sqrt{2d-1} + O(1/\log_d n)$ for any $2d$-regular graph. Therefore, no family of $2d$-regular graphs can have smaller asymptotic $\lambda(G)$ bounds than Theorem 2.

As a consequence of Theorem 2, $\mathbb{H}$-graphs are expanders with $O(\log_d n)$ diameter with high probability because of the relation between eigenvalues and expanders [13], [14].

Corollary 3 is a direct consequence of Theorem 2.

Corollary 3: Let G be a graph chosen from $\mathbb{H}_{n,2d}$ uniformly at random. Then $\lambda(G) \leq 2\sqrt{2d}$ with probability $1 - O(n^{-p})$, where p depends on d.

IV. Perfect Sampling

In this section, we will discuss several implementations for procedure Sample of Section III.

A. Global Server

In a simple centralized solution using a publicly-known sampling server, each joining node can obtain d uniformly random nodes from the sampling server. Each Join or Leave operation only takes $O(1)$ time and $O(1)$ messages. The space required at the sampling server is $O(n)$.

B. Broadcast

Instead of using a central server, a new node u can broadcast a request to all the nodes. Each node will be asked to reply with certain probability $p = O(1/n)$. Thus u expects to receive $O(1)$ replies and can randomly pick one as the sample. A broadcast on an $\mathbb{H}$-graph sends at most $(2d-1)n$ messages and terminates in $\lfloor 2\log_{d/2}(n-1) \rfloor + 1$ steps with high probability. In the worst case, the broadcasting source has to handle $O(n)$ replies. Instead of asking each node to reply independently, a protocol can converge the sampling results at the internal nodes of the broadcast tree, so that each node only need to handle $O(d)$ messages in the worst case. We omit the details of the algorithm in this conference paper.

An efficient implementation of this approach requires an estimation of n. Estimation algorithms are discussed in Section VII-B.

C. Coupling From The Past

Perfect sampling algorithms for Markov chains have been discovered in recent years. Propp and Wilson [15] introduced faster algorithms based on their 'coupling from the past' (CFTP) procedure. They gave an algorithm COVER-CFTP which generates a state distributed with stationary π in expected time at most 15 times the cover time. The expected cover time for $\mathbb{H}$-graphs is around $n \log n$ [16] with high probability. Thus the running time of COVER-CFTP on an $\mathbb{H}$-graph is $O(n \log n)$.

V. APPROXIMATE SAMPLING

All the sampling algorithms in Section IV require $O(n)$ messages. We believe that it is unlikely that perfect sampling with $o(n)$ message complexity is possible for graphs other than the highly-structured ones such as the hypercube. In this section, we describe an approach of approximate sampling using random walks. A graph can be considered as a Markov chain. Sampling by random walks is usually called Markov chain Monte Carlo [17]. Since $\mathbb{H}$-graphs are regular, the limiting distribution of random walks on $\mathbb{H}$-graphs is the uniform distribution. Moreover, since an $\mathbb{H}$-graph is an expander with high probability, a random walk of $O(\log n)$ steps on an $\mathbb{H}$-graph can sample the nodes of the graph with a distribution close to be uniform. We shall show that our protocol with approximate sampling increases the probability of producing 'bad graphs' (those with large $\lambda(G)$) by only a constant factor. Therefore, if the graph is sufficiently large, the high probability result of Theorem 2 can still be applied. With approximate sampling, we need to start with a uniformly distributed probability space of sufficiently large graphs. When the graph is small, we can use a perfect sampling algorithm described in Section IV.

Procedure SAMPLE-RW(t) performs a random walk on the graph and returns the node after t steps.

```
SAMPLE-RW(t)
1  if t = 0
2     then return self
3     else v ← a random element in N(self)
4          return v ⇒ SAMPLE-RW(t − 1)
```

For clarify, SAMPLE-RW is presented as a tail recursion through remote procedure calls. We can as well pass along the address of the initial node. Procedure SAMPLE-RW can replace SAMPLE in Section III. The variant of JOIN using SAMPLE-RW will be denoted as JOIN-RW. Table I summarizes the complexities of the sampling algorithms we have considered.

A. Nodes Joining

We first consider the case where only JOIN-RW is allowed. Lemma 4 is a consequence of Theorem 5.1 in [18].

Lemma 4: Let $G \in \mathbb{H}_{n,2d}$ be a graph such that $\lambda(G) \le 2\sqrt{2d}$, where $d \ge 4$. Let $t = \left\lceil 2\log_{d/2} \frac{n^r}{c} \right\rceil + 4$ where r and c are positive constants. Then for any $v \in G$, we have

$$\left| \Pr\{\text{SAMPLE-RW}(t) \text{ returns } v\} - \frac{1}{n} \right| \le \frac{c}{dn^r}.$$

Theorem 5 shows that although our protocol with SAMPLE-RW does not sample perfectly, the probability that we obtain a graph of large second eigenvalue remains very small.

Theorem 5: Let $\mathcal{S}_n, \mathcal{S}_{n+1}, \mathcal{S}_{n+2}, \ldots$ be a sequence of probability spaces where $\mathcal{S}_k = (\mathbb{H}_{k,2d}, \Sigma_k, P_k)$. Let $\mathcal{S}_n$ be a uniformly distributed probability space and let $\mathcal{S}_{k+1}$ be formed from $\mathcal{S}_k$ by operation JOIN-RW using SAMPLE-RW$\left(\left\lceil 2\log_{d/2} \frac{k^r}{c} \right\rceil + 4\right)$ with $c > 0$ and $r > 2$. Then for all $k \ge n$,

$$P_k\left\{\left\{G \in \mathbb{H}_{k,2d} \mid \lambda(G) \le 2\sqrt{2d}\right\}\right\} \ge 1 - O(n^{-p}),$$

where d is chosen such that $p > 1$.

Proof: Let $\Upsilon_n = \left\{ G \in \mathbb{H}_{n,2d} \mid \lambda(G) \le 2\sqrt{2d} \right\}$ be the set of 'good' graphs in $\mathbb{H}_{n,2d}$. Let $J(G)$ be the set of graphs obtained by operation JOIN (it does not matter whether we consider JOIN or JOIN-RW) on G. For any $k \ge n$, let

$$\Upsilon_{k+1} = \left\{ G \in J(\Upsilon_k) \mid \lambda(G) \le 2\sqrt{2d} \right\}. \tag{1}$$

The probability space $\mathcal{S}_k$ can be considered as a product space $(\mathcal{C}_k)^d$ where $\Omega[\mathcal{C}_k] = \mathbb{H}_k$.

For any $C \in \mathbb{H}_k$ and $l \in \{1, \ldots, d\}$, let $H_k[l, C] = \{ G \in \mathbb{H}_{k,2d} \mid C \text{ is the level-}l \text{ Hamilton cycle of } G \}$ be the set of graphs whose level-l cycle is C.

For any probability space (Ω, Σ, P) and any sets $A, B \subseteq \Omega$, let $P\{A \mid B\} = P\{A \cap B\} / P\{B\}$. We shall prove that for $m \ge n$,

$$P_m\{\Upsilon_m\} \ge 1 - \sum_{j=n}^{m} e^{c\sum_{i=n}^{j-1} i^{1-r}} O(j^{-p}), \text{ and} \tag{2}$$

$$\sup_{\substack{C \in \mathbb{H}_m \\ 1 \le l \le d}} P_m\{ H_m[l, C] \mid \Upsilon_m \} \le \frac{e^{\frac{c}{d}\sum_{i=n}^{m-1} i^{1-r}}}{(m-1)!/2}. \tag{3}$$

Because of our assumption that $\mathcal{S}_n$ is a uniformly distributed probability space and Corollary 3, $P_n\{\Upsilon_n\} \ge 1 - O(n^{-p})$. The eigenvalue of a graph depends on the structure of the graph but not the labels. In other words, a graph has the same eigenvalue no matter how we label the nodes. Thus, we have

$$P_n\{ H_n[l, C] \mid \Upsilon_n \} = P_n\{H_n[l, C]\} = \frac{1}{(n-1)!/2}.$$

Therefore, Inequalities (2) and (3) are satisfied for the base case $m = n$.

Now we assume that Inequalities (2) and (3) are satisfied for $m = k$. We will show that they are satisfied for $m = k+1$.

Let $\mathbf{t}(k, d) = \left\lceil 2\log_{d/2} \frac{k^r}{c} \right\rceil + 4$ be the number of steps walked by SAMPLE-RW.

The set $J(\Upsilon_k)$ are those graphs that can be produced by an operation JOIN on the graphs in Υ_k. Let $C' \in \mathbb{H}_k$ be the cycle such that node $k+1$ is removed from a cycle $C \in \mathbb{H}_{k+1}$.

TABLE I

THE COMPLEXITIES OF THE SAMPLING ALGORITHMS. "SPACE" IS THE WORST-CASE STORAGE REQUIREMENT AT ANY NODE.

Sampling Algorithm	Time	Messages	Space	Sampling
Global Server	$O(1)$	$O(1)$	$O(n)$	perfect
Broadcast	$O(\log_d n)$	$O(dn)$	$O(d)$	perfect
CFTP	$O(n \log n)$	$O(n \log n)$	$O(d)$	perfect
Random Walk	$O(\log_d n)$	$O(\log_d n)$	$O(d)$	approximate

A graph in $H_{k+1}[l, C] \cap J(\Upsilon_k)$ must be created by inserting the node $k+1$ at level l between two nodes of a graph in $H_k[l, C'] \cap \Upsilon_k$. We have

$$\begin{aligned}
&\sup_{C \in \mathbb{H}_{k+1},\, 1 \le l \le d} P_{k+1}\{\, H_{k+1}[l, C] \mid J(\Upsilon_k)\,\} \\
&\le \sup_{C' \in \mathbb{H}_k,\, 1 \le l \le d} P_k\{\, H_k[l, C'] \mid \Upsilon_k\,\} \times \\
&\quad \sup_{v \in [k], G \in \Upsilon_k} \Pr\{\textsc{Sample-RW}(\mathbf{t}(k, d)) \text{ on } G \text{ returns } v\}.
\end{aligned}$$

Let $\sup_{v \in [k], G \in \Upsilon_k} \Pr\{\textsc{Sample-RW}(\mathbf{t}(k, d)) \text{ on } G \text{ returns } v\} = 1/k + \epsilon_k$. Informally, ϵ_k is the amount of deviation resulted from the approximate sampling on some $G \in \Upsilon_k$ by a random walk. It would be zero if a perfect sampling algorithm is used. By Lemma 4, $\epsilon_k \le \frac{c}{dk^r}$. By the inductive assumption of Inequality (3) when $m = k$, we have

$$\begin{aligned}
&\sup_{C \in \mathbb{H}_{k+1},\, 1 \le l \le d} P_{k+1}\{\, H_{k+1}[l, C] \mid J(\Upsilon_k)\,\} \\
&\le \frac{e^{\frac{c}{d}\sum_{i=n}^{k-1} i^{1-r}}}{(k-1)!/2} \frac{1 + \frac{c}{dk^{r-1}}}{k} \\
&\le \frac{e^{\frac{c}{d}\sum_{i=n}^{(k+1)-1} i^{1-r}}}{((k+1)-1)!/2}.
\end{aligned}$$

Since all pairs (C, l) are symmetric, we have

$$\begin{aligned}
&\sup_{C \in \mathbb{H}_{k+1},\, 1 \le l \le d} P_{k+1}\{\, H_{k+1}[l, C] \mid \Upsilon_{k+1}\,\} \\
&\qquad = \sup_{C \in \mathbb{H}_{k+1},\, 1 \le l \le d} P_{k+1}\{\, H_{k+1}[l, C] \mid J(\Upsilon_k)\,\}.
\end{aligned}$$

Thus Inequality (3) is true for $m = k+1$.

Since a new node is inserted into the d Hamilton cycles independently during each JOIN operation, we have

$$\begin{aligned}
&\sup_{G \in J(\Upsilon_k)} P_{k+1}\{\, G \mid J(\Upsilon_k)\,\} \\
&\le \prod_{1 \le l \le d} \sup_{C \in \mathbb{H}_{k+1}} P_{k+1}\{\, H_{k+1}[l, C] \mid J(\Upsilon_k)\,\} \\
&\le \left(\frac{e^{\frac{c}{d}\sum_{i=n}^{(k+1)-1} i^{1-r}}}{((k+1)-1)!/2} \right)^d.
\end{aligned}$$

Dividing both sides by $U_{\mathbb{H}_{k+1,2d}}\{G\}$, we have

$$\sup_{G \in J(\Upsilon_k)} \frac{P_{k+1}\{\, G \mid J(\Upsilon_k)\,\}}{U_{\mathbb{H}_{k+1,2d}}\{G\}} \le e^{c\sum_{i=n}^{(k+1)-1} i^{1-r}}. \quad (4)$$

Starting from Equation (1), we have

$$\begin{aligned}
&P_{k+1}\{\Upsilon_{k+1}\} \\
&= P_{k+1}\left\{\left\{\, G \in J(k) \mid \lambda(G) \le 2\sqrt{2d}\,\right\}\right\} \\
&= \sum_{G \in J(\Upsilon_k),\, \lambda(G) \le 2\sqrt{2d}} P_{k+1}\{G\} \\
&= \sum_{G \in J(\Upsilon_k)} P_{k+1}\{G\} - \sum_{G \in J(\Upsilon_k),\, \lambda(G) > 2\sqrt{2d}} P_{k+1}\{G\}.
\end{aligned}$$

The first term can be derived from our inductive assumption that Inequality (2) is satisfied for $m = k$:

$$\begin{aligned}
\sum_{G \in J(\Upsilon_k)} P_{k+1}\{G\} &= P_{k+1}\{J(\Upsilon_k)\} \\
&= P_k\{\Upsilon_k\} \\
&= 1 - \sum_{j=n}^{k} e^{c\sum_{i=n}^{j-1} i^{1-r}} O(j^{-p}). \quad (5)
\end{aligned}$$

Given Corollary 3 and Inequality (4), the second term can be bounded as follows:

$$\begin{aligned}
&\sum_{G \in J(\Upsilon_k),\, \lambda(G) > 2\sqrt{2d}} P_{k+1}\{G\} \\
&\le \sum_{G \in J(\Upsilon_k),\, \lambda(G) > 2\sqrt{2d}} P_{k+1}\{\, G \mid J(\Upsilon_k)\,\} \\
&\le \sum_{G \in J(\Upsilon_k), \lambda(G) > 2\sqrt{2d}} e^{c\sum_{i=n}^{(k+1)-1} i^{1-r}} U_{\mathbb{H}_{k+1,2d}}\{G\} \\
&< e^{c\sum_{i=n}^{k} i^{1-r}} U_{\mathbb{H}_{k+1,2d}}\left\{\left\{\, G \in \mathbb{H}_{k+1,2d} \mid \lambda(G) > 2\sqrt{2d}\,\right\}\right\} \\
&< e^{c\sum_{i=n}^{k} i^{1-r}} O((k+1)^{-p}). \quad (6)
\end{aligned}$$

Combining Equations (5) and (6), we obtain

$$P_{k+1}\{\Upsilon_{k+1}\} \ge 1 - \sum_{j=n}^{k+1} \left(e^{c\sum_{i=n}^{j-1} i^{1-r}} \right) O(j^{-p}).$$

Since $r > 2$, $e^{c\sum_{i=n}^{k} i^{1-r}}$ is bounded by a constant. Similarly, $\sum_{j=n}^{k} j^{-p}$ is bounded by a constant if $p > 1$. In fact, $\sum_{j=n}^{k} j^{-p}$ becomes negligible quickly with moderate p. For example, when $p = 4$ and $n = 50$, we have $\sum_{j=50}^{\infty} j^{-4} <$

3×10^{-6}. Therefore, for any $k \geq n$,

$$\begin{aligned} & P_k\left\{\left\{G \in \mathbb{H}_{k,2d} \mid \lambda(G) \leq 2\sqrt{2d}\right\}\right\} \\ & \geq P_k\{\Upsilon_k\} \\ & \geq 1 - \sum_{j=n}^{k} e^{c\sum_{i=n}^{j-1} i^{1-r}} O(j^{-p}) \\ & = 1 - O(n^{-p}). \end{aligned}$$

■

We note that Theorem 5 is useful only when the initial graph size n is sufficiently large for $O(n^{-p})$ to be close to zero. We can use complete graphs when the size n is small.

B. Nodes Leaving

The operation LEAVE can have a much larger effect on the probability of obtaining bad graphs. We can obtain a similar result if the number of operations is bounded by a power of the size n of the graphs in the initial uniformly distributed probability space. The proof of Theorem 6 is similar to that of Theorem 5.

Theorem 6: Consider a sequence $\mathcal{S}_0, \ldots, \mathcal{S}_N$ of probability spaces and an integer n such that

- $\langle \sigma_1, \sigma_2, \ldots, \sigma_N \rangle$ is a sequence of operations, such that each operation σ_i is either a LEAVE or a JOIN-RW using SAMPLE-RW($\left\lceil 2\log_{d/2} \frac{k^r}{c} \right\rceil + 4$), where $c > 0$ and $r > 2$.
- $\mathcal{S}_0$ is a uniformly distributed probability space and $\Omega[\mathcal{S}_0] = \mathbb{H}_{n,2d}$;
- $\mathcal{S}_i$ is obtained from $\mathcal{S}_{i-1}$ by operation σ_i for $i = 1, \ldots, N$;
- $|\Omega[\mathcal{S}_i]| \geq n$ for all $i \in \{0, \ldots, N\}$.

If $N = O(n^q)$ such that $q < r - 1$, then

$$P_i\left\{\left\{G \in \Omega[\mathcal{S}_i] \mid \lambda(G) \leq 2\sqrt{2d}\right\}\right\} \geq 1 - O(n^{q-p})$$

for $i = 1, \ldots, N$ and some $p > 1$.

In the long run, the probability spaces may deviate further and further away from the uniformly distributed spaces. (We expect that, in practice, the 'degradation' will be very slow, especially if the leaving nodes are not chosen by an adversary.) In the following, we outline two strategies for slowing down the deviation. When the request sequence is unbounded and arbitrary, we can also use a regeneration algorithm (Section VII-A) to refresh the probability space.

1) The Youngest Node: We observe that if the node to leave is always the node just joined (the youngest node), then Theorem 5 would remain valid. Of course we cannot avoid other nodes leaving before the youngest node z. Nevertheless, when some node v wants to leave, it can just swap its set of neighbors $N(v)$ with the neighbors of z. Thus, for the graph topology, it would appear as if the youngest node z has left instead. We call the ordered list of the neighbors $\langle \mathsf{nbr}_{-1}, \mathsf{nbr}_1, \mathsf{nbr}_{-2}, \ldots, \mathsf{nbr}_{-d}, \mathsf{nbr}_d \rangle$ the *shell* of a node. Swapping shells takes $O(1)$ time and $O(d)$ messages.

Now our problem reduces to locating the youngest node z by any node that wants to leave. Let g be a server known to all nodes in the graph. Each node $u \in G$ contains a field prev that points to the node that joined just before u. Server g's variable youngest should always point to the current youngest node.

- When the youngest node z leaves, $g \Rightarrow$ youngest is updated to $z \Rightarrow$ prev.
- When a new node u joins the network, $(u \Rightarrow \mathsf{prev}) \leftarrow (g \Rightarrow \mathsf{youngest})$ and then $g \Rightarrow$ youngest is updated to u.

Only $O(1)$ messages are required for each JOIN-RW or LEAVE operation. We note that only $O(1)$ space is required at the global server.

2) Shell Recycling: The previous approach either requires a global server or a broadcast during every JOIN-RW or LEAVE. We now investigate a distributed solution without broadcasting.

When a node v departs, it can ask a neighbor u to store v's shell. Then u will simulate node v and have (at most) $4d$ effective neighbors. We call u a *shell-host*. The idea is to save the shell and wait for a new node to adopt it. When there are sufficient shell-hosts, a new node can easily find a shell-host and adopt a shell.

When the set of shell-hosts occupies a proportion ψ of the graph, a new node will only need $O(\psi^{-1})$ expected time to find a shell-host. In the worst case that the set of shell-hosts is determined by an adversary, it will still only take $O(\psi^{-1} + \log n)$ expected time to locate a shell-host.

We note that this will only have a limited effect on the performance of the algorithms running on the graph. If we allow b extra shells for each shell-host, then a graph of size n will be simulating a graph of size $(b+1)n$ in the worst case. Each node has to store $(2d)b$ additional addresses. Since the time complexity of JOIN-RW is $O(\log n)$, the extra time cost is only $O(\log b)$ plus the cost of finding a shell-host.

Although this approach is distributed and efficient, it has the limitation that shell-hosts can become saturated in the extreme case. The number of shells per shell-host dictates the maximum factor that a graph can shrink. For example, if we set the load factor b to be 2, then the entire graph can host at most $2n$ additional shells. This means that a graph reducing its size by two-third will run out of shell-hosts. In this case, we will need a regeneration algorithm (Section VII-A).

At last, we note that it could be beneficial to pre-build some shells so that a new joining node can simply adopt a shell, especially if an expensive perfect sampling algorithm is used.

VI. SIMULATION RESULTS

We have performed some simulations to gain more insights into the constructions of the $\mathbb{H}$-graphs. We would like to obtain some guidelines for setting the various parameters of our algorithms. For example, we would like to know what should be the sufficient size of d for an $\mathbb{H}$-graph of certain size.

According to Theorem 2, if an $\mathbb{H}$-graph G is chosen from a set $\mathbb{H}_{n,2d}$ uniformly at random, then for any $\epsilon > 0$,

$$\lambda(G) \leq 2\sqrt{2d-1} + \epsilon \tag{7}$$

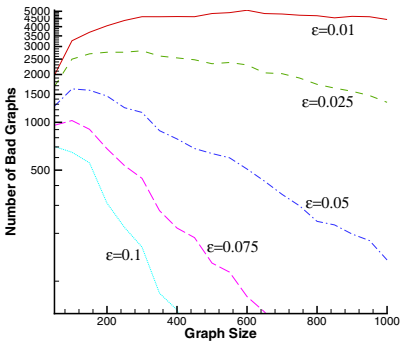

Fig. 2

NUMBER OF GRAPHS NOT SATISFYING INEQUALITY (7) IN 100,000 TRIALS, FOR $d = 4$ AND $\epsilon = 0.01, 0.025, 0.05, 0.075, 0.1$.

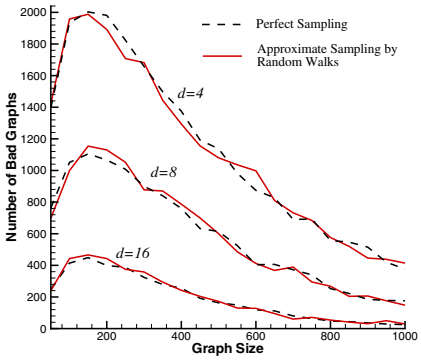

Fig. 3

NUMBER OF GRAPHS NOT SATISFYING INEQUALITY (7) IN 100,000 TRIALS, FOR $d = 4, 8, 16$ AND $\epsilon = d/100$.

with probability $1-O(n^{-p})$, where p depends on d. However, it is not clear that given ϵ, what size of n is sufficient so that $O(n^{-p})$ is insignificant.

Our program first creates a 3-node $\mathbb{H}$-graph and then have new nodes joined sequentially. When the size of the graph is $50, 100, \ldots, 1000$, we find the second largest eigenvalue of the graph by calling ARPACK++ [19], an eigenvalue package.

We simulate our construction algorithm using sampling algorithm SAMPLE-RW. As suggested by Lemma 4, number of steps taken is set to $\left\lceil 2\log_{d/2}\frac{n^r}{c}\right\rceil + 4$. We fixed $c = 1$ and $r = 3$ (no observable effects on the results when r is set to 1 or 5). Since the purpose of the simulations is to evaluate the second largest eigenvalues of the graphs produced by our protocol, we used the graph size n, instead of an estimate (discussed in Section VII-B), for determining the number of random walk steps.

Let us call a graph 'bad' if it does not satisfy Inequality (7). In our simulations, we count the number of bad graphs observed in 100,000 independent trials. In Figure 2, with $d = 4$, we can see that the number of bad graphs drops quicker against increasing graph size when the ϵ in Inequality (7) is larger. For example, when $\epsilon = 0.1$, we observed 218 bad graphs when $n = 250$, 26 bad graphs when $n = 500$, and 0 bad graph when $n = 1000$. Our results in Section V are based on Corollary 3 that takes $\epsilon = 2(\sqrt{2d} - \sqrt{2d-1})$, which is around 0.37 for $d = 4$. With this ϵ value, only 20 graphs of size 50 were bad. No bad graphs were observed for any graph of size at least 100 in the 100,000 trials.

Next, we investigated the effects of parameter d (half of the node degree). The solid lines in Figure 3 represent the number of bad graphs for $d = 4, 8, 16$, with $\epsilon = d/100$. The ϵ is set to be proportional to d because the eigenvalues should be normalized by dividing $2d$, the largest eigenvalue, for a fair comparison. The number of bad graphs decrease with increasing d. The dashed lines represent the graphs using perfect sampling. We did not observe a significant difference between the results using perfect sampling and the results using approximated sampling.

According to our simulation results, $d = 4$ is sufficient to obtain a high probability of satisfying Inequality (7) for $\epsilon = 2(\sqrt{2d} - \sqrt{2d-1})$. A larger d will give us smaller eigenvalues for small graphs, lower time complexity for joining and diameter (both $O(\log_d n)$), but higher space and message complexities.

The high probability results in Section V only apply when we start with a uniform probability space of sufficient graph size. However, the simulations show that even if we start from the 3-node $\mathbb{H}$-graph, there is a small probability of obtaining bad graphs in the long run, as most of the bad graphs become 'good' when the graph size increases.

VII. MAINTENANCE ALGORITHMS

A. Regeneration

By various reasons, the probability space produced by our protocol may deviate too far away from the uniformly distributed space. It could be caused by the extraordinary shrinkage discussed in Section V-B.2, or by some node failures. Although we are unable to find a distributed algorithm to 'repair' a probability space, we can regenerate the graph by creating a new set of Hamilton cycles. In the following, we will present a regeneration algorithm.

Our approach is to start from a small graph again and insert nodes until the new graph has taken in all existing nodes. Apparently, it would take up to n steps to construct a new $\mathbb{H}_{n,2d}$ graph. However, it is possible to speed up this process by joining nodes in parallel. For example, consider a set of new nodes joining G at the same time so that some node $v \in G$ is picked simultaneously by k new nodes at level i. In this case, v needs to insert the k nodes between v and $v \Rightarrow \mathsf{nbr}_i$ in a random order.

During each round, we will let each node in the new graph invite its neighbors in the old graph to join. For example, let v_0, v_1, v_2 be the initial members of the new graph. In the first round, they will invite all their neighbors to join the new graph.

Let these neighbors be $v_3, \ldots, v_{k_1}$. In the second round, after nodes $v_3, \ldots, v_{k_1}$ have joined, they can in turn invite their neighbors to join. At every step, the number of new nodes is at most $2d-1$ times the current size of the new graph. Therefore, each existing node in the new graph expects to see $O(d^2)$ LINK requests. Effectively, this process expands like a broadcast, which takes $O(\log_d n)$ rounds to cover the entire graph.

We can use a perfect sampling algorithm when the graph is small and switch to SAMPLE-RW after some size n_0. The overall time and message complexities are $O(\log_d^2 n)$ and $O(d(n_0^2 + n\log_d))$.

We can invoke a regeneration based on a fixed schedule or an estimate of the number of LEAVE operations.

B. Size Estimation

In a distributed setting, any node in a graph would not know the total size of the graph. However, most of our algorithms, like sampling with broadcasts (Section IV-B) or random walks (Section V), would run more efficiently if we have a good estimation of the graph size. Therefore, we would like to have an efficient distributed algorithm to gather and disseminate estimates of the graph size.

Estimating the size of a graph distributedly is an interesting algorithmic problem. Our goal is for every node to have a lower-bound estimate on the graph size. Since JOIN-RW has time complexity $O(\log n)$, if our estimate of the graph size is n^μ, $\mu \geq 1$, then the complexity is increased by a factor of μ. An estimate can be obtained with a central server or with a broadcast tree, with algorithms similar to corresponding ones in Section IV. We note that although size estimation is not cheap, the cost can be amortized over many join and leave operations. We discuss two estimation protocols in the following.

a) Random Walk: Feige [20] gave a randomized LOGSPACE algorithm COMPONENT to estimate the size of an arbitrary connected graph in $O(n^3)$ time and a regular graph in $O(n^2)$ time. COMPONENT was used as a subroutine for testing graph connectivity. For expander graphs, we can apply Gillman [21]'s Chernoff bound for random walks. Adapting Gillman's modified Aldous's procedure [22], we can show that, in $O(\beta^2 n \log n)$ time and with $O(\beta^2 n)$ messages, a node can estimate the size of the graph within error βn with high probability.

b) Covering Walk: We are currently working on a new distributed estimation algorithm based on long-lived random walkers on the graph. A walker counts the number of nodes that it has visited by marking the nodes. It also notifies the nodes with its current count.

VIII. LAYERED $\mathbb{H}$-GRAPHS

An $\mathbb{H}$-graph is efficient for locating a member of a subset, when the size of the subset is not too small when compared with the size of the entire graph (see Section IX-A). However, if we want to locate a particular node, the expected time is $\Theta(n)$. In this section, we will improve the performance for such application by overlaying multiple $\mathbb{H}$-graphs on the same set of nodes.

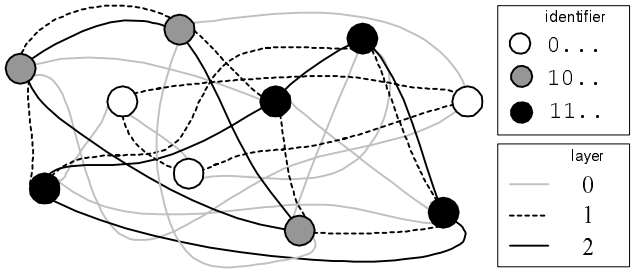

Fig. 4
A LAYERED $\mathbb{H}$-GRAPH WITH $d = 1$.

Let us start with two layers. Let G be connected as an $\mathbb{H}$-graph by our protocol. We can randomly color half of the nodes red and half of the nodes blue. All the red nodes are then connected by an $\mathbb{H}$-graph consisting of red nodes only. And there is an $\mathbb{H}$-graph for the blue nodes. In this case, if we want to find a particular red node v, we first find any red node using the original $\mathbb{H}$-graph, and then locate node v following the edges of the red $\mathbb{H}$-graph.

The next natural step is to consider having layers of $\mathbb{H}$-graphs until the deepest layer consists of graphs of $O(1)$ size. We call such topology a layered $\mathbb{H}$-graph. Let layer 0 refers to the $\mathbb{H}$-graph connecting the entire graph.

We will assign an identifier to each node. Let an identifier be a string $z_1 z_2 \cdots z_m$ such that $z_i \in \{0, \ldots, q_i - 1\}$ for all $i \in \{1, \ldots, m\}$. We will assume that $q_i = q$ in this paper, although our results can be extended to the general case of distinct q_i's. Let $\mathsf{id}(u)$ be any node u's identifier.

We first describe how identifiers can be used to construct a layered $\mathbb{H}$-graph. Consider a set of n nodes such that each node is assigned an identifier. The nodes are connected by *layers* of $\mathbb{H}$-graphs. In layer 0, the nodes are connected as a standard $\mathbb{H}$-graph. Then for each layer i and each prefix $z_1 \cdots z_i$, the nodes with this prefix are connected as a $\mathbb{H}$-graph (if there are at least three such nodes). For example, nodes with $z_1 = 0$ should be connected as an $\mathbb{H}$-graph, and all nodes with $z_1 = 1$ should be connected as a separate $\mathbb{H}$-graph. Figure 4 shows an example of a 3-layer $\mathbb{H}$-graph.

To search for a node z in a layered $\mathbb{H}$-graph, we first search for a node u with $\mathsf{id}(u)_1 = z_1$ using the layer-0 links. After that, we can use the layer-1 links of the z_1-subgraph to search for a node v with $\mathsf{id}(v)_2 = z_2$. We can repeat this process until we reach a subset of $O(1)$ size, which can be connected as a complete graph.

For any node u in a layered $\mathbb{H}$-graph G, let $\mathsf{layer}(u, i) = (V', E')$, such that

$$V' = \{w \in V(G) \mid w \text{ can reach } u \text{ with layer-}i \text{ edges}\},$$
$$E' = \{(x, y) \mid x, y \in V' \text{ and } (x, y) \text{ is in layer } j \geq i\}.$$

Let $|\mathsf{layer}(u,i)|$ be the number of nodes in $\mathsf{layer}(u,i)$. Our protocol will maintain the property that if $v \in \mathsf{layer}(u,i)$, then $\mathsf{id}(v)_h = \mathsf{id}(u)_h$ for $0 \le h < i$, i.e., v and u share the same $(i-1)$-prefix.

The number of layers of a graph does not need to be the length of the identifiers. When the graph is small, few layers are sufficient. Let $\mathsf{basesize} > d$ be a constant such that for any $G = \mathsf{layer}(u,i)$,

- if $|G| < \mathsf{basesize}$, then G is a complete graph and does not contain deeper layers;
- if $|G| > 2 \times \mathsf{basesize}$, then G is a layered $\mathbb{H}$-graph and contains at least one deeper layer.

Transitions (procedures ToComplete and ToRegular) between a complete graph and an $\mathbb{H}$-graph are not complicated and are omitted in this conference paper.

If $\mathsf{layer}(u,i)$ is a complete graph, then $\mathsf{layer}(u,j) = (\{u\}, \emptyset)$ for $j > i$.

```
IdSearch(z, i)
1  return a node u such that id(u)_i = z_i
```

Procedure IdSearch can be implemented by walking along a Hamilton cycle. It can shown that the expected time complexity is the inverse of the proportion of nodes satisfying $\mathsf{id}(u)_i = z_i$. We omit the details of the algorithm in this conference paper.

Now we present the algorithms for joining and leaving a layered $\mathbb{H}$-graph.

```
LayerJoin(u, i)
1  if layer(self, i) is a complete graph
2     then insert u into the complete graph
3          if |layer(self, i)| > 2 × basesize
4             then ToRegular(i)
5     else Join(u, i)
6          IdSearch(id(u), i + 1) ⇒ LayerJoin(u, i + 1)
```

For any node u in a layered $\mathbb{H}$-graph G, let $\mathsf{depth}(u)$ be the deepest layer that it belongs to. Let $\mathsf{depth}(G)$ be the maximum depth of the nodes in G.

```
LayerLeave(i)
1  if |layer(self, depth(self) − 1)| < basesize
2     then ToComplete(depth(self) − 1)
3  for each j ∈ {i, ..., depth(self)} in parallel
4  do Leave(j)
```

Procedures $\textsc{Join}(u,i)$ and $\textsc{Leave}(i)$ are like the original Join and Leave except that they only use the edges in layer i.

The LayerSearch algorithm moves recursively from layer 0 to the deepest layer of the target node.

```
LayerSearch(z, i)
1  if id(self) is a prefix of z
2     then return self
3     else return IdSearch(z, i + 1) ⇒ LayerSearch(z, i + 1)
```

The major advantage of layered $\mathbb{H}$-graph over plain $\mathbb{H}$-graph is that any node can be located efficiently given its identifier.

Given a layered $\mathbb{H}$-graph G and a node $u \in G$. We say that a non-complete-graph $\mathsf{layer}(u,i)$ is *δ-balanced* if for any $v \in \mathsf{layer}(u,i)$,

$$\frac{|\mathsf{layer}(v,i+1)|}{|\mathsf{layer}(u,i)|} \ge \frac{\delta}{q}.$$

If $\mathsf{layer}(u,i)$ is δ-balanced for all $u \in G$ and all $i \in \{0,\ldots,\mathsf{depth}(G)\}$ where $\mathsf{layer}(u,i)$ is not a complete graph, then we say G is δ-balanced.

Corollary 7: If G is δ-balanced, then LayerSearch takes at most $(q/\delta - 1)\log_q n$ hops in expectation.

The best possible expected number of hops is $\log_2 n$ on a 1-balanced layered $\mathbb{H}$-graph when $q = 2$.

As indicated by Corollary 7, it is crucial that the layered $\mathbb{H}$-graph is balanced. We will consider two approaches of assigning the identifiers.

A. Random Identifiers

In this subsection we assume that identifiers are chosen uniformly at random. A hash function can be used to map the name of a node into an identifier z. Alternatively, each new node can pick an identifier randomly. We will also assume that the identifiers have sufficient number of bits.

One can expect that a uniform hash function or a good random number generator should be able to produce layered $\mathbb{H}$-graphs with good bounded δ. We will verify this intuition in the following.

We can show that layered $\mathbb{H}$-graphs of sufficient size are likely to be 1/2-balanced, and prove Theorems 8 and 9, which give the time complexities of LayerJoin and LayerSearch.

Theorem 8: Let G be a layered $\mathbb{H}$-graph with uniformly distributed identifiers. With high probability, the expected time complexity of $\textsc{LayerJoin}(u,0)$ is $O(\log_q n + \log_d n)$.

Theorem 9: If the identifiers of a layered $\mathbb{H}$-graph G are randomly distributed, then with high probability (with respect to the topology G), $\textsc{LayerSearch}(z,0)$

- returns in expected times $O(q \log n)$, and
- returns in $O(q \log^2 n)$ steps with high probability (with respect to the random choices during the search).

B. Sampling Identifiers

Sometimes a solution based on random or hashed identifiers is unsatisfactory because of the potential collisions and the uncertainty when n is small. Moreover, even though we can assume that joining nodes have uniformly distributed identifiers, if the leaving nodes are picked by an adversary, then the graph can become unbalanced.

For certain applications, we can construct a layered $\mathbb{H}$-graph that is "self-correcting" by allowing the new nodes to choose their own identifiers with the goal of enhancing the balancedness of the graph.

We will illustrate the approach for the case that $q = 2$. The idea is to balance the tree by trying to join the smaller subset. For example, in a graph with 1000 nodes, if there are 700 nodes with $z_1 = 0$ and 300 nodes with $z_1 = 1$, a new node joining the graph should try to set its z_1 to 1. It is very expensive to actually count the number of nodes with $z_1 = 0$ or $z_1 = 1$. However, it is not difficult for a node to guess intelligently which set is larger. If node u samples one node in the graph, there is a probability of 7/10 observing a $z_1 = 0$ node and a probability of 3/10 observing a $z_1 = 1$

node. So if u observes a $z_1 = 0$ node, then it should set its z_1 to 1, and vice versa. For a more prudent decision, u can sample more nodes and pick the one with the least-frequently observed prefix. This method can be generalized to arbitrary q:

```
SAMPLING-IDENTIFIER(q, i)
1  u ← SAMPLE(i − 1) // using the layer-(i − 1) edges only
2  return a random number in {0, ..., q − 1} \ id(u)_i
```

For example, we can consider a 2-layer $\mathbb{H}$-graph. For any q, let $S(q)_n$ be the number of $z_1 = 0$ nodes in a layered $\mathbb{H}$-graph of n nodes is constructed with SAMPLING-IDENTIFIER$(q, 1)$. Let random variable $X(q)_n$ be the number of $z_1 = 0$ nodes when identifiers chosen uniformly at random.

The analyses of $S(q)_n$ are omitted in this conference paper. The following summary of results, comparing $S(q)_n$ with the binomially distributed variable $X(q)_n$, confirms our intuition that sampling identifiers should lead to better load-balancing.

$$\mathrm{E}\left[X(q)_n\right] = n/q \qquad \mathrm{E}\left[S(q)_n\right] = n/q$$

$$\mathrm{Var}\left[X(q)_n\right] = \frac{(q-1)n}{q^2} \qquad \mathrm{Var}\left[S(q)_n\right] \approx \frac{(q-1)^2 n}{q^2(q+1)}$$

$$\mathrm{E}\left[e^{tX(2)_n}\right] = \left(\frac{e^t+1}{2}\right)^n \qquad \mathrm{E}\left[e^{tS(2)_n}\right] < \left(\frac{e^t-1}{t}\right)^n$$

$$\Pr\left\{X(2)_n < \frac{n}{4}\right\} < (0.883)^n \qquad \Pr\left\{S(2)_n < \frac{n}{4}\right\} < (0.665)^n$$

IX. Applications

A. Community Discovery

In many situations, we want to locate a subset of nodes sharing some common interests with us. For example, consider the following scenarios:

- Alice likes to locate other anime fans of a certain studio, so that she can share unlicensed video clips with them.
- Bob wants to discover some peer-to-peer servers storing open-sourced software packages.

$\mathbb{H}$-graphs can be used to discover such communities. Let A be a subset of nodes in an $\mathbb{H}$-graph G such that $|A| / |G| = \psi$. For any such set A, we can find a member of A in $O(\log_d \psi^{-1})$ time with $O(d\psi^{-3})$ messages with high probability by a broadcast. Moreover, we can find a member of A by random walk in $O(\psi^{-1} + \log n)$ hops in expectation even if set A is selected by an adversary. Analyses are omitted in this conference paper.

We suggest that $\mathbb{H}$-graph can be used to establish a large global network with potentially millions of nodes, in which smaller communities can be formed and searched easily. These communities can be connected as smaller $\mathbb{H}$-graphs or by other domain-specific protocols.

B. Lookup Service

A layered $\mathbb{H}$-graph can be used to implement a distributed lookup service similar to those supported by CAN [1], Chord [2], Pastry [3], and Tapestry [4]. A lookup service stores key-value pairs in the network, such that the values can be looked up efficiently when given the key. To implement a lookup service on the layered $\mathbb{H}$-graph, keys can be hashed into identifiers. A key is stored at the node whose identifier has the longest prefix match with the hashed identifier. Insertions, deletions, and updates of entries have the same time and message complexities as procedure LAYERSEARCH. In particular, the $O(\log n)$ routing time and $O(\log n)$ neighbor size of layered $\mathbb{H}$-graphs match the corresponding complexities in these four systems.

Thus the randomness of these topologies solely depends on the hash functions used. On the other hand, the randomness of $\mathbb{H}$-graphs arises from the independent random walks (in addition to the hash function). As a consequence, $\mathbb{H}$-graphs are tolerant to an adversary selecting the leaving nodes.

$\mathbb{H}$-graphs are particularly efficient for node departures because a leaving node only needs to notify its immediate neighbors. In other systems, some re-organization is necessary. In addition, layered $\mathbb{H}$-graphs support a 'self-balancing' technique by sampling the existing identifiers. This leads to better load-balancing.

C. Dynamic Connectivity

Layered $\mathbb{H}$-graph can implement a community where the underlying network address of any node may change from time to time. For example, a node connected to the Internet by DHCP can have its IP address changed every few days. On a laptop or PDA with wireless access, the IP address may change frequently. A layered $\mathbb{H}$-graph allows us to form such dynamic networked community (for chatting, gaming, file-sharing, etc.) without using a centralized directory. Any node can reach another node in $O(\log n)$ hops given the identifier. Layered $\mathbb{H}$-graph is particularly suitable for such application because a) a node changing its IP address only needs to notify $O(\log n)$ neighbors in $O(1)$ time, and b) the network is tolerant to a large number of nodes temporarily unavailable.

X. Related Work

Little has been done on distributed algorithms for random graphs. Frieze and McDiarmid [23]'s survey on random graphs algorithms included several parallel algorithms but no distributed algorithms. There were several sequential algorithms for generating random regular graphs [24], [25], [26]. Random graphs constructed by centralized algorithms were also used for communication networks [27], [28].

Kermarrec, Massoulié, and Ganesh [29] and Eugster et al. [30] have studied randomized networks for probabilistic broadcasts and membership management. The analysis in [30] is based on the assumption each node having "uniformly distributed random view" of a constant size, justified by some simulation results. It is not clear how fast their graphs converge to a topology satisfying the assumption. In [29], the random views of the nodes are provided by a set of dedicated servers that require $O(n)$ storage space.

Pandurangan, Raghavan, and Upfal [31] proposed an algorithm for building low-diameter peer-to-peer networks with bounded degrees. Although their random topology also achieves $O(\log n)$ diameter with high probability, their approach and techniques are different from ours. First, they

assume a stochastic model for the arrivals and departures of nodes, while $\mathbb{H}$-graph assumes arbitrary sequence of arrivals and departures. Second, their protocol uses a special server that is known to all nodes in the network, which is not required in $\mathbb{H}$-graphs. Third, their network is connected with high probability instead of with certainty.

XI. Concluding Remarks

This paper introduces a distributed approach for constructing overlay networks as random graphs. Using random walks as a sampling subroutine, we have demonstrated a scalable construction of expander graphs without any centralized server support.

Previous studies of random regular graphs indicate that they are robust against failures [32], [33]. We expect that most isolated faults would have limited effects on the probability spaces. Further work is required to devise efficient schemes for recovering from node failures and schedules for the regeneration algorithm.

In a large network, there can be a large variation of network distance (which can be a function of the network delay, bandwidth, or reliability) between pairs of nodes. We can have a hierarchy of $\mathbb{H}$-graphs based on the underlying network topology. For example, there can be an $\mathbb{H}$-graph for each regional network, an $\mathbb{H}$-graph for each country, and then a global $\mathbb{H}$-graph. Each search query will first try to search within the smaller $\mathbb{H}$-graphs before querying nodes in the larger $\mathbb{H}$-graphs.

In a heterogeneous network, a powerful machine can serve as several nodes in the (layered) $\mathbb{H}$-graphs, thus increasing the number of its effective neighbors by a constant factor. Such scheme effectively utilizes machines of different capacities and is transparent to our algorithms.

At last, we expect that the distributed sampling algorithms and estimation algorithms discussed in this paper could lead to interesting further investigations, and be useful in other situations.

References

[1] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker, "A scalable content addressable network," in *Proceedings of SIGCOMM 2001*, Aug. 2001, pp. 161–172.

[2] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan, "Chord: A scalable peer-to-peer lookup service for internet applications," in *Proceedings of SIGCOMM 2001*, Aug. 2001, pp. 149–160.

[3] A. Rowstron and P. Druschel, "Pastry: Scalable, decentralized object location and routing for large-scale peer-to-peer systems," in *Proceedings of the 18th IFIP/ACM International Conference on Distributed Systems Platforms*.

[4] B. Y. Zhao, J. Kubiatowicz, and A. Joseph, "Tapestry: An infrastructure for fault-tolerant wide-area location and routing," University of California, Berkeley, Tech. Rep. CSD-01-1141, 2001.

[5] A. Frieze, M. Jerrum, M. Molloy, R. Robinson, and N. Wormald, "Generating and counting Hamilton cycles in random regular graphs," *Journal of Algorithms*, vol. 21, no. 1, pp. 176–198, July 1996.

[6] A. M. Frieze and L. Zhao, "Optimal construction of edge-disjoint paths in random regular graphs," in *Proceedings of the Tenth Annual ACM-SIAM Symposium on Discrete Algorithms*, Jan. 17–19 1999, pp. 346–355.

[7] H. Garmo, "The asymptotic distribution of long cycles in random regular graphs," *Random Structures and Algorithms*, vol. 15, 1999.

[8] M. Krivelevich, B. Sudakov, V. H. Vu, and N. C. Wormald, "Random regular graphs of high degree," *Random Structures and Algorithms*, vol. 18, 2001.

[9] J. Friedman, "On the second eigenvalue and random walks in random d-regular graphs," *Combinatorica*, vol. 11, pp. 331–362, 1991.

[10] M. Harchol-Balter, T. Leighton, and D. Lewin, "Resource discovery in distributed networks," in *Proceedings of the 18th ACM Symposium on Principles of Distributed Computing*, May 1999.

[11] B. Bollobás, *Random Graphs*, 2nd ed. Cambridge University Press, Sept. 2001.

[12] J. Friedman, "A proof of Alon's second eigenvalue conjecture," 2002. [Online]. Available: http://www.math.ubc.ca/~jf/pubs/index.html

[13] N. Alon, "Eigenvalues and expanders," *Combinatorica*, vol. 6, no. 2, pp. 83–96, 1986.

[14] N. Kahale, "Eigenvalues and expansion of regular graphs," vol. 42, no. 5, pp. 1091–1106, Sept. 1995.

[15] J. G. Propp and D. B. Wilson, "How to get a perfectly random sample from a generic Markov chain and generate a random spanning tree of a directed graph," *J. Algorithms*, vol. 27, no. 2, pp. 170–217, May 1998.

[16] D. J. Aldous and J. A. Fill, "Reversible Markov chains and random walks on graphs," monograph in preparation.

[17] M. Jerrum and A. Sinclair, "The Markov chain Monte Carlo method: an approach to approximate counting and integration," in *Approximation Algorithms for NP-hard Problems*, D. S. Hochbaum, Ed. PWS Publishing, 1996, pp. 482–520.

[18] L. Lovász, "Random walks on graphs: a survey," *Combinatorics, Paul Erdős is Eighty (vol. 2)*, pp. 353–398, 1996.

[19] F. M. Gomes and D. C. Sorensen, "ARPACK++: A C++ implementation of ARPACK eigenvalue package." CRPC, Rice University, Houston, TX, Tech. Rep. TR97729, 1997.

[20] U. Feige, "A fast randomized LOGSPACE algorithm for graph connectivity," *Theoretical Computer Science*, vol. 169, no. 2, pp. 147–160, Dec. 1996.

[21] D. Gillman, "A Chernoff bound for random walks on expander graphs," *SIAM Journal on Computing*, vol. 27, no. 4, pp. 1203–1220, Aug. 1998. [Online]. Available: http://epubs.siam.org/sam-bin/dbq/article/26876

[22] D. Aldous, "On the Markov chain simulation method for uniform combinatorial distributions and simulated annealing," *Probability in the Engineering and Informational Sciences*, vol. 1, pp. 33–46, 1987.

[23] A. Frieze and C. McDiarmid, "Algorithmic theory of random graphs," *Random Structures & Algorithms*, vol. 10, pp. 5–42, 1997.

[24] B. McKay and N. C. Wormald, "Uniform generation of random graphs of moderate degree," *Journal of Algorithms*, vol. 11, pp. 52–67, 1990.

[25] A. Steger and N. C. Wormald, "Generating random regular graphs quickly," *Combinatorics, Probability and Computing*, vol. 8, pp. 377–396, 1999.

[26] G. Katona and A. Seress, "Greedy construction of nearly regular graphs," *European Journal of Combinatorics*, vol. 14, 1993.

[27] Farago, Chlamtac, and Basagni, "Virtual path network topology optimization using random graphs," in *Proceedings of the 1999 IEEE Computer and Communications Societies Conference on Computer Communications*, 1999.

[28] A. Srinivasan, K. G. Ramakrishnan, K. Kumaran, M. Aravamudan, and S. Naqvi, "Optimal design of signaling networks for Internet telephony," in *Proceedings of the 2000 IEEE Computer and Communications Societies Conference on Computer Communications*, Mar. 2000, pp. 688–716.

[29] A.-M. Kermarrec, L. Massoulié, and A. Ganesh, "Reliable probabilistic communication in large-scale information dissemination systems," Microsoft Research Cambridge, Tech. Rep. 2000-105, Oct. 2000.

[30] P. Eugster, S. Handurukande, R. Guerraoui, A.-M. Kermarrec, and P. Kouznetsov, "Lightweight probabilistic broadcast," in *Proceedings of The International Conference on Dependable Systems and Networks*, July 2001.

[31] G. Pandurangan, P. Raghavan, and E. Upfal, "Building low-diameter P2P networks," in *Proceedings of the 42nd Annual IEEE Symposium on the Foundations of Computer Science*, Oct. 2001.

[32] N. C. Wormald, "The asymptotic connectivity of labelled regular graphs," *Journal of Combinatorial Theory (B)*, vol. 31, pp. 156–167, 1981.

[33] S. E. Nikoletseas, K. V. Palem, P. G. Spirakis, and M. Yung, "Connectivity properties in random regular graphs with edge faults," *International Journal of Foundations of Computer Science*, vol. 11, 2000.