Time- and Communication-Efficient Overlay Network Construction via Gossip

Fabien Dufoulon
School of Computing and Communications, Lancaster University, UK

Michael Moorman
Department of Computer Science, University of Houston, TX, USA

William K. Moses Jr.
Department of Computer Science, Durham University, UK

Gopal Pandurangan
Department of Computer Science, University of Houston, TX, USA

Abstract

We focus on the well-studied problem of distributed overlay network construction. We consider a synchronous *gossip-based* communication model where in each round a node can send a message of small size to another node whose identifier it knows. The network is assumed to be *reconfigurable*, i.e., a node can add new connections (edges) to other nodes whose identifier it knows or drop existing connections. Each node initially has only knowledge of its own identifier and the identifiers of its neighbors. The overlay construction problem is, given an arbitrary (connected) graph, to reconfigure it to obtain a bounded-degree *expander* graph as efficiently as possible. The overlay construction problem is relevant to building real-world peer-to-peer network topologies that have desirable properties such as low diameter, high conductance, robustness to adversarial deletions, etc.

Our main result is that we show that starting from any *arbitrary* (connected) graph G on n nodes and m edges, we can construct an overlay network that is a constant-degree *expander* in polylog n rounds using only $\tilde{O}(n)$ messages.[1] Our time and message bounds are *both* essentially optimal (up to polylogarithmic factors). Our distributed overlay construction protocol is very lightweight as it uses gossip (each node communicates with only one neighbor in each round) and also scalable as it uses only $\tilde{O}(n)$ messages, which is *sublinear* in m (even when m is moderately dense). To the best of our knowledge, this is the first result that achieves overlay network construction in polylog n rounds and $o(m)$ messages. Our protocol uses graph sketches in a novel way to construct an expander overlay that is both time and communication efficient.

A consequence of our overlay construction protocol is that distributed computation can be performed very efficiently in this model. In particular, a wide range of fundamental tasks such as broadcast, leader election, and minimum spanning tree (MST) construction can be accomplished in polylog n rounds and $\tilde{O}(n)$ message complexity in any graph.

2012 ACM Subject Classification Theory of computation → Distributed algorithms; Mathematics of computing → Probabilistic algorithms; Mathematics of computing → Discrete mathematics

Keywords and phrases Peer-to-Peer Networks, Overlay Construction Protocol, Gossip, Expanders, Sublinear Bounds

Digital Object Identifier 10.4230/LIPIcs.ITCS.2024.42

Related Version *Full Version*: https://arxiv.org/abs/2311.17115

Funding *Fabien Dufoulon*: Part of the work was done while Fabien Dufoulon was a postdoctoral fellow at the University of Houston, Houston, TX, USA. F. Dufoulon was supported in part by National Science Foundation (NSF) grants CCF-1540512, IIS-1633720, and CCF-1717075 and U.S.-Israel Binational Science Foundation (BSF) grant 2016419.

[1] The notation $\tilde{O}$ hides a polylog n multiplicative factor.

15th Innovations in Theoretical Computer Science Conference (ITCS 2024).
Editor: Venkatesan Guruswami; Article No. 42; pp. 42:1–42:23
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Michael Moorman: M. Moorman was supported in part by NSF IIS-1633720 REU Supplement grant.
William K. Moses Jr.: Part of the work was done while William K. Moses Jr. was a postdoctoral fellow at the University of Houston, Houston, TX, USA. W. K. Moses Jr. was supported in part by NSF grants CCF-1540512, IIS-1633720, and CCF-1717075 and BSF grant 2016419.
Gopal Pandurangan: G. Pandurangan was supported in part by NSF grants CCF-1540512, IIS-1633720, and CCF-1717075 and BSF grant 2016419.

1 Introduction

1.1 Background and Prior Work

Many of today's large-scale distributed systems in the Internet are peer-to-peer (P2P) or overlay networks. In such networks, the (direct) connections between nodes can be considered as *virtual (or logical)* connections as they make use of the physical connections of the underlying Internet. Furthermore, in these networks, a node can communicate with another node if it knows the IP address of the other node, and can also (potentially) establish a connection (link) to it. Thus, the network can *reconfigure* itself, choosing which connections to add and which to drop.

In this paper, we consider the well-studied problem of constructing an efficient overlay topology in a distributed fashion in a reconfigurable network. Overlay construction is particularly important in modern P2P networks, which depend significantly on topological properties to ensure efficient performance. In fact, over the last two decades, several theoretical works [44, 39, 27, 17, 17, 32, 11] have focused on building P2P networks with various desirable properties such as high conductance, low diameter, and robustness to a large number of adversarial deletions. The high-level idea in all these works is to distributively build a (bounded-degree) *random graph* topology which guarantees the above properties. This idea exploits the fact that a random graph is an *expander* with high probability and hence has all the above desirable properties [31, 43].[2] Indeed, random graphs have been used extensively to model P2P networks (see e.g., [39, 44, 27, 17, 40, 12, 8, 9, 7]). It should also be noted that the random connectivity topology is widely deployed in many P2P systems today, including those that implement blockchains and cryptocurrencies, e.g., Bitcoin [41].

Several prior works [3, 26, 30] have addressed the problem of constructing an expander topology starting from an *arbitrary* topology network. One of the earliest works is that of Angluin et al. [3] who showed how one can transform an arbitrary connected graph G on n nodes and m edges into a binary search tree of depth $O(\log n)$ in $O(d + \log n)$ rounds and $O(n(d + \log n))$ messages, where d is the maximum degree. (Their model is similar to ours, where a node can only send a message to a single neighbor per round.) It can be shown that an $O(\log n)$-depth binary tree can be transformed into many other desirable topologies (such as an expander, butterfly, or hypercube). The work of Gilbert et al. [26] presented a distributed protocol that when given any (connected) network topology having n nodes and m edges will transform it in to a given (desired) target topology such as an expander, hypercube, or Chord, with high probability. This protocol incurred $O(\operatorname{polylog} n)$ rounds and exchanged messages of only small size ($O(\log n)$ bits) per communication link per round, and had a total message complexity of $\tilde{\Theta}(m)$.[3] Note that while the protocol of Gilbert et al. has a better time complexity compared to Angluin et al., when d is large, the protocol of Gilbert et al. is not gossip-based (i.e., a node can send messages to all its neighbors in one round, even if its degree is large), unlike that of Angluin et al.

[2] Throughout, "with high probability (w.h.p.)" means with probability at least $1 - 1/n^c$ for some constant $c \geqslant 1$; n is the network size.

[3] The notation $\tilde{O}$ hides a $\operatorname{polylog} n$ multiplicative factor.

The recent work of Götte et al. [30] presented an overlay construction algorithm that when given an arbitrary (connected) graph, transforms the graph into a *well-formed tree*, i.e., a rooted tree of constant degree and $O(\log n)$ diameter. In particular, their protocol first constructs an $O(\log n)$-degree expander (a well-formed tree can be obtained from the expander by known techniques). The protocol takes $O(\log n)$ rounds which is asymptotically time-optimal, since $\Omega(\log n)$ rounds is a lower bound for constructing a well-formed tree or a constant-degree expander from an arbitrary graph [30]. However, their protocol takes $\tilde{\Theta}(m)$ messages (m is the number of edges in the starting graph) as each node needs to send $d \log n$ messages in a round where d is the initial (maximum) degree. The novelty of their protocol is the repeated use of short (constant length) random walks to increase the graph conductance.

We note that all the above overlay construction protocols, while being fast, i.e., taking only $O(\text{polylog}\, n)$ rounds, use $\tilde{\Theta}(m)$ messages, i.e., linear in the number of edges of the initial graph. An important question is whether one can design overlay construction protocols that are significantly communication-efficient, i.e., taking $o(m)$ messages or even $\tilde{O}(n)$ messages. In fact, the work of Götte et al. [30] raised the question of whether it is possible to obtain a fast overlay construction protocol that is also communication-efficient.

In this paper, we answer the above question and present the first overlay construction protocol that is both time- and communication-efficient: given an arbitrary connected graph on n nodes and m edges, the protocol constructs a constant-degree expander in $\text{polylog}\, n$ rounds using only $\tilde{O}(n)$ messages (regardless of the value of m). We note that our protocol uses only messages of small size ($O(\text{polylog}\, n)$ bits). Furthermore, it uses *gossip-based communication* which is fully-decentralized and lightweight. Hence, it inherits the usual advantages of gossip-based protocols (e.g., see [35, 19, 34]) such as robustness, no single point of action, etc.[4]

1.2 Model

Before we formally state our main result, we discuss our model which is similar to that used in previous work on overlay network construction (see e.g., [3]).

We assume that we are given a connected *arbitrary* graph $G = (V, E)$ as input. Let $|V| = n$ and $|E| = m$. Each node has a unique ID (identifier) taken from the range $[1, N]$, where $N = n^c$ for some positive constant $c \geqslant 1$. Thus, each node ID can be represented using $O(\log n)$ bits.

We assume that the communication links are *reconfigurable*: if a node u knows about the ID of some node v, then u can establish or drop a link to v.[5] Also, as is standard in overlay (and P2P) networks, a node can communicate with another node if it knows the identity of the other node.

The computation proceeds in *synchronous* rounds and the communication topology produced by the execution evolves as a sequence of graphs $G = G_1, G_2, \ldots$, where $G_r = (V, E_r)$ corresponds to the network at the beginning of round r. The graph G_1 is the initial configuration, and determines the initial knowledge of nodes which is restricted to only knowledge of their own ID and the IDs of their (respective) neighbors. This is a standard model in distributed computing, called the *Knowledge-Till-Radius 1* ($KT1$) model. Note

[4] For this reason, we avoid more centralized methods such as building and using a BFS tree for aggregation, etc., in favor of fully-decentralized gossip-based aggregation protocols [35, 19].

[5] Strictly speaking, it takes a successful handshake between u and v to establish or drop a bidirectional link. For simplicity, and since it does not change the asymptotic bounds of our results, we assume that these connections happen instantaneously.

that in overlay and P2P networks, this model is the natural model, as every node knows the identity (IP address) of itself and of its neighbors. Nodes initially have no knowledge of any other nodes (other than their respective neighbors) or any global knowledge including the initial topology. However, we assume that nodes know an upper bound on n (in fact, just a constant factor upper bound of $\log n$ is sufficient).

We assume a *gossip-based* communication model which is very lightweight, where in a round, a node can send a small-sized message (of size $O(\text{polylog}\, n)$ bits) to only one of its neighbors. Thus, a round r consists of the following three steps: (i) each node contacts one neighbor, (ii) each node sends a $O(\text{polylog}\, n)$ bit message to the contacted neighbor and receives a $O(\text{polylog}\, n)$ bit reply,[6] and (iii) after all messages in transit have been received, u performs some local computation possibly including changes to its communication links, resulting in G_{r+1}. We call this model the P2P-GOSSIP model. Note that although the gossip model allows each node to send a message to only one neighbor per round, one can easily simulate sending messages to any constant $k \geqslant 1$ neighbors in a round, by performing gossip for k rounds and thus blowing up the number of rounds by a factor of k. On the other hand, in the standard CONGEST model, a node can send a message (of small size, say $O(\log n)$ bits) to all its neighbors in a round. In addition, in the LOCAL model, the message size is unbounded.

1.3 Our Contributions

Main Result. Our main contribution is a distributed protocol for overlay network construction that given an *arbitrary* connected graph, constructs an overlay graph whose topology is a constant-degree *expander*. Informally, an expander is a graph that has *constant* conductance (see definition in Section 2.1). As mentioned earlier, an expander graph has very desirable properties: low diameter ($O(\log n)$), high conductance, fast mixing of random walks (i.e., a random walk reaches stationary distribution in $O(\log n)$ rounds, which is useful for fast random sampling), and robustness to large adversarial deletions (deleting even a constant fraction of nodes leaves a giant component of size $\Theta(n)$ that is also an expander) [31, 13, 10, 9].

The protocol takes $\text{polylog}\, n$ rounds and uses only $\tilde{O}(n)$ messages.[7] These time and message bounds are *both* essentially optimal (up to polylogarithmic factors), since it is easy to show that $\Omega(\log n)$ is a lower bound on time [30] and $\Omega(n)$ is a lower bound on the number of connections that need to be added/deleted (and hence the number of messages).[8]

Our distributed overlay construction protocol (Section 4) is very lightweight and scalable, as it uses gossip-based communication (each node communicates with only one neighbor per round). To the best of our knowledge, this is the first result that achieves overlay network construction in $O(\text{polylog}\, n)$ rounds and $o(m)$ messages, i.e., sublinear in m, the number of edges of the initial graph. All prior protocols took at least $\tilde{O}(m)$ messages in general while taking $\text{polylog}\, n$ rounds. We note that once an expander topology is constructed, several other well-known topologies such as hypercube, butterfly, binary tree, etc. can be constructed [30, 3].

[6] Note that although each node sends a message to only one neighbor, a node can receive as many messages as its degree in a round (e.g., the center node in a star graph). We assume that the node replies to all them in the round.

[7] We have not chosen to optimize the log factors in our protocol, as this was not the primary focus. As it is, our protocol takes $O(\log^5 n)$ rounds and this can be improved.

[8] Consider a dumbbell graph consisting of two cliques joined by a single edge as the starting graph. To convert this graph to a constant degree expander, at least $\Theta(n)$ edges have to be added between the cliques, and the cliques themselves have to be sparsified by dropping all but a constant number of random edges.

We simulated our protocol to study its performance in several types of graphs (Section 5). The results validate the theory and shows that in all the different types of graphs, the conductance increases significantly to essentially the best possible. The algorithm also finishes fast, i.e., in a few phases.

Implications. A consequence of our overlay construction protocol, is that distributed computation can be performed very efficiently in the P2P-GOSSIP model. In particular, a wide range of fundamental tasks such as broadcast, leader election, and spanning tree (ST) construction can be accomplished in $\text{polylog}\, n$ round complexity and $O(n \,\text{polylog}\, n)$ message complexity in any graph. This follows because one can first construct a constant-degree expander using the overlay construction protocol and then do the above tasks on the expander graph (which has $O(\log n)$ diameter and $O(n)$ edges) in $O(\log n)$ rounds and $\tilde{O}(n)$ messages by just simulating standard CONGEST model algorithms in gossip [48, 38].[9]

Furthermore, one can also show that the minimum spanning tree (MST) problem can be solved very efficiently. (Note that in the MST problem, we are given an arbitrary (connected) undirected graph G with edge weights, and the goal is to find the MST of G.) We will outline how this can be accomplished in $\text{polylog}\, n$ rounds and $O(n \,\text{polylog}\, n)$ messages which is a consequence of this work and prior works. First, using our expander overlay protocol, we add a (constant-degree) expander overlay on G (i.e., the expander edges are added in addition to the edges of G). The expander edges will be used for efficient communication in G. For this, we convert the expander (that is not addressable) into an hypercube (that is addressable) which allows efficient routing between any two nodes in $O(\log n)$ rounds and $O(n \log n)$ messages. This conversion can be accomplished using the techniques of [3, 28, 4] (see also [30]) or the protocol of [5].[10] All these protocols take $\text{polylog}\, n$ rounds and $\tilde{O}(n)$ messages to convert a constant-degree expander into an hypercube. Using the addressable hypercube overlay on top of G, we can efficiently implement the Gallagher-Humblet-Spira (GHS) algorithm [22] as shown in Chatterjee et al. [16] to compute the MST of G in $\text{polylog}\, n$ rounds and $\tilde{O}(n)$ message complexity.[11]

We note that the above results are a significant improvement in the round complexity of solving the above fundamental problems in the standard $KT1$ model. King, Kutten, and Thorup [37] showed that all these problems can be solved using $\tilde{O}(n)$ messages (regardless of the value of m, the number of edges of the graph), but this takes $\tilde{O}(n)$ rounds in the standard CONGEST model. Obtaining sublinear, i.e., $\tilde{O}(n)$ messages, in time that is significantly faster than $O(n)$ is an open problem in the CONGEST model (see also [42, 29, 37]). In contrast, we show that in the P2P-GOSSIP model, one can solve these problems in sublinear ($\tilde{O}(n)$) messages and $\text{polylog}\, n$ time.

Another implication is to the complexity of the GOSSIP model, see e.g., [15, 14] and the references therein. The works of Censor-Hillel et al. [14] studied the complexity of distributed computation in the GOSSIP model – where a node may only initiate contact

[9] Since the graph is of constant degree, one can easily simulate a round of the CONGEST model, where a node sends a message to all its neighbors, by performing gossip for a constant number of rounds.

[10] The techniques of [3, 4, 30] construct a well-formed tree which can then be transformed into an hypercube. The well-formed tree can be easily constructed from an expander [30]. Alternatively, one can use the protocol of [5] which gives a fully-decentralized and robust protocol using random walks (which can be simulated using gossip) to construct an addressable hypercube from a constant-degree expander.

[11] We note that Chatterjee et al. uses the permutation routing algorithm of Ghaffari and Li [24] (also see [23]) which takes $2^{O(\sqrt{\log n})}$ rounds on an expander (which is not addressable). In contrast, exploiting the reconfigurability of the P2P model, we can *reconfigure* the expander into an hypercube which allows permutation routing to be accomplished in $O(\log n)$ rounds using the standard Valiant routing [49].

with a single neighbor in each round, but *unbounded* messages sizes are allowed (unlike in our P2P-GOSSIP model which uses small message sizes) – in comparison to the more standard model of distributed computation, namely, the much less restrictive LOCAL model, where a node may simultaneously communicate with all of its neighbors in a single round (also message sizes can be unbounded). This work studied the complexity of the *information dissemination* problem in which each node has some (possibly unique) data item and each node is required to collect all the data items from all nodes. They gave an algorithm that solves the information dissemination problem in at most $O(D + \operatorname{polylog} n)$ rounds in a network of size n and diameter D. This is at most an additive polylogarithmic factor from the trivial lower bound of D, which applies even in the LOCAL model. In fact, they prove that any algorithm that requires T rounds in the LOCAL model can be simulated in $O(T + \operatorname{polylog} n)$ rounds in the GOSSIP model, showing that GOSSIP and LOCAL models are essentially equivalent (up to polylogarithmic factors). Our work shows that in the P2P-GOSSIP model, if we allow unbounded message sizes, one can solve information dissemination in $O(\operatorname{polylog} n)$ rounds in a straightforward way by first constructing a constant-degree expander (that has diameter $O(\log n)$) and then doing gossip on the expander [15].

High-level Overview and Technical Contributions. Our overlay construction protocol (Section 4) uses a combination of several techniques in a non-trivial way to construct an expander overlay in $\tilde{O}(n)$ messages and $\operatorname{polylog} n$ time.

Our expander overlay construction protocol is conceptually simple and is similar to the classic GHS algorithm [22] and consists of several phases. In the first phase, we start with n clusters, each corresponding to a node. In a phase, adjacent clusters are merged and the protocol maintains the invariant that each cluster is a *constant-degree expander* at the end of each phase. (More precisely, there exists some "uniform" constant δ such that any cluster in any phase is an expander of degree at most δ.)

We now describe a phase of the protocol which consists of three major steps. By the maintained invariant, we can assume that all the clusters are constant-degree expanders (this is trivially true in the first phase, since each cluster is a singleton node). As in GHS, one has to quickly find outgoing edges from each cluster. Except for the first phase, it is non-trivial to accomplish this using gossip in $\operatorname{polylog} n$ rounds and using $O(n \operatorname{polylog} n)$ messages per phase. The first major step in a phase is to efficiently aggregate sketches from each cluster. Informally, a sketch of a node is a short representation of its adjacency list, i.e., using $O(\operatorname{polylog} n)$ bits (see Section 2.2). An important property of these graph sketches is that the sketches of all nodes in a cluster can be aggregated in $O(\operatorname{polylog} n)$ bits and an outgoing edge of the cluster can be found with high probability from the aggregated sketch. A main technical contribution, that can be of independent interest, is showing how one can use graph sketches [1, 2] to efficiently sample an outgoing edge using gossip in an expander. We show that we can adapt the PUSH-SUM gossip protocol of [35] to efficiently aggregate sketches in an accurate manner in all nodes of a cluster (Section 3.2).

In the second step, each cluster uses the aggregated sketch to sample an outgoing edge. The clusters along with their respective outgoing edges induce a disjoint set of connected components. Each such component has to be converted into a constant degree expander. To accomplish this, we use the protocol of Götte et al. [30] that takes an arbitrary *constant-degree* graph and converts it into an expander that has $O(\log n)$ degree.[12] One cannot directly

[12] Note that the protocol of [30] works on graphs of somewhat higher degree, say $O(\log n)$, but the maximum degree needs to be small to get the performance guarantees as claimed.

invoke the protocol of [30] on each connected component, since the degree of a node may not be constant (multiple outgoing edges can go to a single node). In the second step, to satisfy the degree bound needed for the protocol of [30] (invoked in step 3) we do a *degree reduction* to keep the degrees of all nodes constant.

In the third step, we run the protocol of [30] which uses random walks (and can be simulated in the gossip model with an $O(\log n)$ factor slow down in the number of rounds) to convert each connected component into an $O(\log n)$-degree expander in polylog n rounds of gossip. We then run an efficient distributed protocol for reducing the degree of the expander to $O(1)$, which may also be of independent interest (Section 3.1). This is essential in maintaining the invariant of the next phase (without this degree reduction, one can show that nodes' degrees can grow large from phase to phase).

We show that each phase reduces the number of clusters by a constant factor and hence the total number of phases is $O(\log n)$. At the end, the whole graph becomes one cluster, which will be a constant degree expander.

1.4 Additional Related Work and Comparison

There have been several prior works on P2P and reconfigurable networks (e.g., [46, 20, 39, 11]) that assume an expander graph to start with, and then faults occur (insertions or deletions of nodes due to churn or other dynamic changes), even repeatedly, and the goal is to maintain the expander. One can view the current paper (as well as prior works discussed earlier [30, 3, 26]) as a preprocessing step that starts with an arbitrary graph and converts it to an expander. Subsequently, one may use the protocols from these works to maintain the expander. For future work, it will be useful to extend our protocol to a dynamic setting where churn is present. In this context, one can see the current work as a first step towards obtaining a protocol that builds and maintains an expander overlay in *dynamic* P2P networks that suffer from churn and where the churn can result in (intermediate) arbitrary topologies that are far from expanders (but still connected, say). In that case, a protocol like ours will be useful to reconstruct an expander.

Finally, we point out that there has been work on other models that are similar, yet different, compared to the P2P-GOSSIP model. Two notable examples include the node-congested clique model [6] and the hybrid model [28]. In the node-congested clique model, each node is constrained to send only a small amount of messages of small size per round, say $O(\log n)$ messages each of size $O(\log n)$ bits. In the hybrid model, two modes of communication are assumed: the LOCAL (or CONGEST) model in the input graph G, where a node can send unlimited (or small-sized) messages on the (local) edges of G and a node-congested clique model where a node can send $O(\log n)$-size messages to $O(\log n)$ other nodes. Note that instead of a clique model (where a node can potentially communicate with any other node), one can also assume an $O(\log n)$-degree expander (or another sparse degree structure such as a tree) built on top of G and nodes can communicate on these using small-sized messages. This variant of the hybrid model is assumed in [30]. A main difference between the P2P-GOSSIP model and the node-congested clique model is that in the latter, it is assumed that each nodes knows the IDs of all other nodes which makes routing trivial. Whereas in the P2P-GOSSIP model, a node knows only the IDs of its neighbors. Also in the node-congested clique model, each node can send or receive only $O(\log n)$ messages, whereas, in the P2P-GOSSIP model, since the graph is arbitrary a node can receive as many messages as its degree. Efficient algorithms for MST (and other problems) that take polylogarithmic rounds are known in the node-congested clique model. In the hybrid model variant of [30], logarithmic round algorithms have been given for several fundamental problems such as spanning trees, MIS, etc. However, it is left open whether an efficient algorithm can be designed for MST in the hybrid model [30].

2 Preliminaries

2.1 Graph Definitions

For any graph $G = (V, E)$, its (maximal) connected components are called *clusters*. Moreover, for any node $v \in V$, the neighbors of v are denoted by $N(v) = \{u \mid (u, v) \in E\}$ and the degree of v by $d(v) = |N(v)|$. The *volume* of any subset $S \subseteq V$ is defined as $vol(S) = \sum_{v \in S} d(v)$, and the *edge cut* of S as $E(S, V \setminus S) = \{(u, v) \mid (u, v) \in E, u \in S, v \notin S\}$. The *conductance* $\Phi(S)$ of any subset $S \subseteq V$ is defined as $\Phi(S) = |E(S, V \setminus S)| / \min\{vol(S), vol(V \setminus S)\}$. The conductance of graph G is defined as $\Phi = \min_{S \subseteq V, S \neq \emptyset} \Phi(S)$. Finally, for any edge set $E' \subseteq V^2$, we say that E' *generates* the graph $G(E') = (V, E')$ is called the *graph generated*. Note that E' may include edges in $V^2 \setminus E$ (i.e., peer-to-peer edges when G is the communication graph) and thus $G(E')$ is not the graph induced by E'.

A family of graphs G_n on n nodes is an *expander family* if, for some constant α with $0 < \alpha < 1$, the conductance $\phi_n = \phi(G_n)$ satisfies $\phi_n \geqslant \alpha$ for all $n \geqslant n_0$ for some $n_0 \in \mathbb{N}$.

2.2 Graph Sketches

Consider an arbitrary set of nodes V, such that all nodes of V have unique IDs in $[1, N]$, where N is known to all nodes. We define, for any graph $G = (V, E)$ and node $v \in V$, the *incidence vector* $\mathbf{i}_G(v) \in \mathbb{R}^{\binom{N}{2}}$ whose entries correspond to all possible choices of two IDs in N. An entry in $\mathbf{i}_G(v)$ corresponding to the possible edge between u with $id_u \in N$ and v is 0 if $(u, v) \notin E$, 1 if $(u, v) \in E$ and $id_u > id_v$, and -1 otherwise (i.e., $(u, v) \in E$ and $id_u < id_v$). Entries in $\mathbf{i}_G(v)$ that correspond to a possible edge not including v (e.g., (u, w)) have value 0. Naturally, one can extend this definition to any node subset $S \subseteq V$: more precisely, $\mathbf{i}_G(S) = \sum_{v \in S} \mathbf{i}_G(v)$. Note that by linearity, the non-zero indices of $\mathbf{i}_G(S)$ indicate exactly which edges are in the cut of S with respect to G, that is, in $E_G(S, V \setminus S)$.

One may use the incidence vector of some node set S to sample an outgoing edge from S, if one exists, uniformly at random from all such outgoing edges, i.e., sample a non-zero entry in $\mathbf{i}_G(S)$ uniformly at random. However, in a distributed setting, to compute the incidence vector on a set of nodes S, one would need to aggregate the incidence vectors of the nodes belonging to S. This is problematic since incidence vectors are exponentially larger (recall that they have size $\binom{N}{2}$) than our $O(\text{polylog}\, N)$ message size.

Fortunately, we can use a *linear sketch* [1] – a well-chosen linear function from $\mathbb{R}^{\binom{N}{2}}$ to $\mathbb{R}^k$ – or in other words, a *graph sketching matrix* – a well-chosen $k \times \binom{N}{2}$ size matrix M_G – to compress these vectors $(\mathbf{i}_G(v))_{v \in V}$ into smaller (sketch) vectors $(\mathbf{s}_G(v))_{v \in V}$ of size $k = O(\text{polylog}\binom{N}{2})$; more concretely, $M_G \cdot \mathbf{i}_G(v) = \mathbf{s}_G(v)$. Moreover, although this compression necessarily loses some information, it has two major advantages. First, it is possible to sample (almost) uniformly at random one of the non-zero indices of $\mathbf{i}_G(v) \in \mathbb{R}^{\binom{N}{2}}$ by performing operations on $\mathbf{s}_G(v)$ only (albeit with some small failure probability). More concretely, for any graph sketching matrix M_G and for any subset $S \subseteq V$, there exists a sampling function f_G that takes the sketch vector $\mathbf{s}_G(S)$ as input and outputs an edge chosen uniformly at random in $E_G(S, V \setminus S)$. Second, the linearity of the graph sketching matrix allows us to compute $\mathbf{s}_G(S)$ without computing $\mathbf{i}_G(S)$, but instead by computing the sketch vectors $(\mathbf{s}_G(v))_{v \in S}$ and summing them.

In summary, for any subset $S \subseteq V$, we can sample an edge chosen uniformly at random in $E_G(S, V \setminus S)$ by (i) having nodes agree on $\Theta(\log^2 n)$ true random bits that they can use to locally compute a common graph sketching matrix M_G with polynomially bounded integer entries [33, 45], (ii) aggregating the sketch vectors of all nodes in S, and (iii) applying the sampling function f_G on the aggregate vector $\mathbf{s}_G(S)$. Note that these steps require messages of small $O(\text{polylog}\, n)$ size only. A more formal statement is given below, which can be obtained straightforwardly by adapting known results [33, 1, 45].

▶ **Lemma 1.** *For any upper bound N on the ID range and constant $0 < \delta < 1$, there exist a graph sketching matrix M_G (with entries polynomially bounded in N) and a sampling function f_G such that for any node subset $S \subset V$, the aggregate sketch vector $\boldsymbol{s}_G(S) = \sum_{u \in S} \boldsymbol{s}_G(u)$ can be represented using $O(\text{polylog}\, n)$ bits, and $f_G(\boldsymbol{s}_G(S))$ samples a (uniformly) random edge in $E_G(S, V \setminus S)$ with probability $1 - \delta$.*

2.3 Creating Expanders from Bounded Degree Graphs

To make our work self-contained, we briefly describe Procedure CreateExpander – the overlay construction algorithm from [30] – and its guarantees here. For any constant $\Phi \in (0, 1/2]$, integer $d \geqslant 1$ and d-bounded degree graph $G = (V, E)$, Procedure CreateExpander first performs some preprocessing on the graph G to transform it into an $O(\log n)$-regular benign graph H_0. (A formal definition of benign graphs is given below, but roughly speaking, these are graphs on which random walks have some good properties.) After which, starting from the edges of H_0, Procedure CreateExpander computes in each phase a new set of edges generating a graph with twice as much conductance – up to $1/2$. After some $O(\log n)$ phases, Procedure CreateExpander terminates and outputs an $O(\log n)$-regular expander graph with conductance Φ.

▶ **Lemma 2** ([30] with an $O(\log n)$ overhead in the P2P-GOSSIP model)**.** *For any constant $\Phi \in (0, 1/2]$, integer $d \geqslant 1$ and d-bounded degree graph $G = (V, E)$, Procedure CreateExpander uses $O(\log^2 n)$ rounds and $O(n \log^2 n)$ messages to output an $O(\log n)$-regular graph $H' = (V, E'_H)$ with conductance Φ.*

Next, we provide more details about the phases of Procedure CreateExpander. First, let $\Delta_H = \Delta_H(d) = \Theta(\log n)$ and $\Lambda = \Theta(\log n)$ be two parameters chosen via the analysis. We call $H = (V, E_H)$, a multi-graph composed of peer-to-peer (multi-)edges, a (Δ_H, Λ)-*benign graph* if it holds that (i) H is Δ_H-regular, that is, every node of H has Δ_H incident (multi-)edges, (ii) H is lazy, that is, every node of H has at least $\Delta_H/2$ self-loops, and (iii) the minimum cut of H is at least Λ. Then, Procedure IncreaseExpansion – implementing one phase of Procedure CreateExpander – is run on a (Δ_H, Λ)-benign graph and outputs a (Δ_H, Λ)-benign graph with better conductance. More precisely, in Procedure IncreaseExpansion, each node initiates $\Delta_H/8$ random walks of length $\ell = \Theta(1)$. These random walks take $\Theta(\log n)$ rounds to terminate (since nodes may only send 1 message per round in the P2P-GOSSIP model). After these $O(\log n)$ rounds, each node generates, for each token it holds, an edge to the token's originating node. If a node holds more than $3\Delta_H/8$ tokens, then that node randomly chooses $3\Delta_H/8$ tokens without replacement and creates edges accordingly. Finally, each node adds self-loops until it has Δ_H incident edges.

▶ **Lemma 3** ([30])**.** *For any (Δ_H, Λ)-benign graph $H = (V, E_H)$ with conductance Φ, Procedure IncreaseExpansion uses $O(\log n)$ rounds and $O(n \log n)$ messages to output a (Δ_H, Λ)-benign graph $H' = (V, E'_H)$ with conductance $\Phi' \geqslant \min\{(\Phi\sqrt{\ell})/640, 1/2\}$. For $\ell \geqslant (2 \cdot 640)^2$, it holds that $\Phi' \geqslant \min\{2\Phi, 1/2\}$.*

3 Our Primitives

In this section, we develop two primitives (that can be of independent interest) that we subsequently use in our main algorithm. The first primitive, in Section 3.1, allows us to modify an $O(\log n)$-regular expander graph into an $O(1)$-bounded degree expander. The second primitive, in Section 3.2, allows one to construct an aggregate sketch vector in an $O(1)$-bounded degree expander using a gossip-style approach, assuming that each node initially contains its own sketch vector.

3.1 Degree Reduction for Expanders

Consider a high conductance $O(\log n)$-regular expander graph G. We give a procedure, called Procedure ExpanderDegreeReduction, that sparsifies this expander graph into a δ-bounded degree expander graph G_δ with any desired conductance $\Phi \in (0, 1/10]$, in $O(\log^3 n)$ rounds and $O(n \log^3 n)$ messages, where $\delta = O(1)$ is some integer determined by the analysis.

Initially, each node generates c active tokens with its ID, where c is a positive integer such that $c < \delta/10$. These tokens will be used to generate the edges of the procedure's resulting graph G_δ. The algorithm works in $O(\log n)$ phases, where the precise value is determined by the analysis. In each phase, active tokens take random walks of length $\ell = O(\log n)$ per phase – where ℓ is larger than the mixing time on G. (More concretely, each node that holds an active token sends the token to a neighbor chosen uniformly at random.[13]) At the end of each phase, any token that ends at some node with at most δ tokens becomes inactive and remains at that node for the remainder of the algorithm. (If during some phase too many tokens end at a node, such that it holds strictly more than δ tokens, then for simplicity all these tokens remain active.) Moreover, for each token that becomes inactive, the node holding it creates a temporary (1-round) edge to inform its source (of the inactive token). A node is said to be *satisfied* once all of its generated tokens are inactive. Once all nodes are satisfied, each satisfied node creates one edge (in G_δ) to each node holding at least one of its inactive tokens.

Analysis. We start with a simple invariant obtained via counting argument (see Lemma 4). With this invariant, we can show that the number of active tokens reduces by half in each phase with constant probability (see Lemma 5). As a result, we can show that all nodes become satisfied within $O(\log n)$ phases (see Lemma 6), or in other words, before the algorithm terminates.

▶ **Lemma 4.** *For any given phase, fix the random walks of all tokens except one. Then, it holds that at least* $0.9n$ *nodes have strictly less than* δ *tokens.*

Proof. We use a simple counting argument to show the lemma statement. Assume by contradiction that more than $n/10$ nodes have more than δ tokens. By the algorithm definition, there can only be $c \cdot n$ tokens in total, and thus $c \cdot n \geqslant \delta \cdot n/10$. However, since $c < \delta/10$, $c \cdot n < \delta \cdot n/10$, which leads to a contradiction. ◀

▶ **Lemma 5.** *In any given phase, the number of active tokens reduces by half with probability at least* $1/2$.

Proof. Let $k \leqslant c \cdot n$ be the initial number of active tokens in this phase. These tokens, denoted by $t_1, \ldots, t_k$, each take an ℓ-length random walk. For any $i \in \{1, \ldots, t\}$, let $\mathcal{V}_i$ be the random variable denoting the node the i-th token ends at. Note that ℓ is chosen sufficiently greater than the mixing time on G, thus the i-th token ends at an (almost) uniform random node. More formally, for any node $v \in V$, $\Pr[\mathcal{V}_i = v] \in [1/n - 1/n^a, 1/n + 1/n^a]$ for some constant $a \geqslant 1$. Then, the following rough upper bound holds: for some constant $\varepsilon \in (0, 1)$, for any node $v \in V$, $\Pr[\mathcal{V}_i = v] \leqslant (1 + \varepsilon)/n$.

[13] We show later that sending these tokens in our model incurs an overhead of $O(\log n)$.

Next, for any token t_i, let $R(t_i) = (X_0, \dots, X_\ell)$ denote the random walk of t_i (i.e., the sequence of nodes visited by token t_i) and let $\mathbb{1}_i$ indicate that t_i ends the phase as active. We use $\bar{R}(t_i)$ as shorthand for $(R(t_1), \dots, R(t_{i-1}), R(t_{i+1}), \dots, R(t_k))$, or in other words, to denote all tokens' random walks except that of token t_i. Note that if we fix all random walks except that of t_i – i.e., if we fix $\bar{R}(t_i)$ – then Lemma 4 implies that there exists a set $G(\bar{R}(t_i))$ of nodes, each with strictly less than δ tokens, and such that $|G(\bar{R}(t_i))| \geqslant 0.9n$. If t_i ends up at any nodes of $G(\bar{R}(t_i))$, it becomes inactive. Thus, $\Pr\left[\mathbb{1}_i = 1 \mid \bar{R}(t_i)\right] \leqslant \Pr\left[\mathcal{V}_i \notin G(\bar{R}(t_i)) \mid \bar{R}(t_i)\right] \leqslant \Pr\left[\bigvee_{v \notin G(\bar{R}(t_i))} \mathcal{V}_i = v \mid \bar{R}(t_i)\right] \leqslant (1+\varepsilon)/10$, where the last inequality is obtained through union bound. Consequently, $\Pr[\mathbb{1}_i = 1] \leqslant (1+\varepsilon)/10$.

Let the random variable $A = \sum_{i=1}^{k} \mathbb{1}_i$ denote the number of active tokens when the phase ends. By the above inequality, $\mathbb{E}[A] = \sum_{i=1}^{k} \Pr[\mathbb{1}_i = 1] \leqslant (1+\varepsilon)k/10$. Finally, Markov's inequality implies the lemma statement: that is, $\Pr[A \geqslant 0.5k] \leqslant \Pr[A \geqslant 2\mathbb{E}[A]] \leqslant 1/2$. ◀

▶ **Lemma 6.** *After $O(\log n)$ phases, all nodes are satisfied.*

Proof. A phase is said to be successful if the number of active tokens reduces by half. By Lemma 5, each phase is successful with probability at least 1/2. A simple application of Chernoff bounds imply that there are at least $\log(c \cdot n)$ successful phases after large enough $O(\log n)$ phases. Consequently, after $O(\log n)$ phases, there remain no active tokens, and hence no unsatisfied nodes. ◀

Now, we can show the correctness of the primitive, and bound its round and message complexities, in Theorem 8.

▶ **Lemma 7.** *For any constant $\Phi \in (0, 1/10]$ and any two integers $n, s \geqslant 1$ such that $s \leqslant n/2$, $\binom{n}{s}\binom{cs}{(1-\Phi)cs}((1+\varepsilon)s/n)^{(1-\Phi)cs} \leqslant 1/n^2$ for large enough n and some suitably chosen integer $c \geqslant 1$ and constant $\varepsilon \geqslant 0$.*

Proof. Let $p^* = \binom{n}{s}\binom{cs}{(1-\Phi)cs}((1+\varepsilon)s/n)^{(1-\Phi)cs}$. We give two upper bounds: the first for $s = o(n)$ and the second for $s = \kappa n$ for some constant $0 < \kappa \leqslant 1/2$.

To get the first bound, we use the inequality $\binom{y}{x} \leqslant (ey/x)^x$, that holds for any integers $y \geqslant x \geqslant 1$. Then, $p^* \leqslant (en/s)^s (e/(1-\Phi))^{(1-\Phi)cs}((1+\varepsilon)s/n)^{(1-\Phi)cs} = 2^{s(\beta + (1-(1-\Phi)c)(\log n - \log s))}$, where $\beta = \log(e) + (1-\Phi)c\log(e(1+\varepsilon)/(1-\Phi))$. For large enough n, the $(1-(1-\Phi)c)\log n$ factor dominates in the exponent. Thus, it suffices to choose c large enough and it holds that $p^* \leqslant 1/n^2$ for large enough n.

To get the second bound, we need to use a tighter inequality (since s and n are large) to bound the binomial coefficients. More concretely, using Stirling's formula, it holds that $\binom{y}{x} \leqslant 2^{yH(x/y)}$, for any integers $x, y \geqslant 0$ and where $H(q) = -q\log(q) - (1-q)\log(1-q)$ is the binary entropy of $q \in (0,1)$. As a result, we get $p^* \leqslant 2^{n(H(\kappa) + c\kappa H(1-\Phi) + (1-\Phi)c\kappa\log((1+\varepsilon)\kappa))}$. First, we take small enough ε such that for any $\kappa \leqslant 1/2$, $|\log((1+\varepsilon)\kappa)| \geqslant 0.9$. Next, note that $H(1-\Phi) \leqslant 2\sqrt{\Phi(1-\Phi)} \leqslant 0.6$, where the first inequality is a well-known upper bound for binary entropy that holds for any $\Phi \in (0,1)$ and the second one holds because $2\sqrt{x(1-x)}$ takes value 0.6 at $x = 1/10$ and increases between $x = 0$ and $x = 1/2$. Then, $H(1-\Phi) \leqslant |(3/4)(1-\Phi)\log((1+\varepsilon)\kappa)|$, since the right-hand side is strictly greater than 0.6 for $\Phi \leqslant 1/10$. Therefore, $p^* \leqslant 2^{n(H(\kappa) + (c\kappa/4)(1-\Phi)\log((1+\varepsilon)\kappa))}$. Next, we take c large enough so that $|(c\kappa/8)(1-\Phi)\log((1+\varepsilon)\kappa)| \geqslant 2|\kappa\log\kappa|$. Then, $p^* \leqslant 2^{n(\kappa\log\kappa - (1-\kappa)\log(1-\kappa) + (c\kappa/8)(1-\Phi)\log((1+\varepsilon)\kappa))}$. Since $\kappa\log\kappa - (1-\kappa)\log(1-\kappa) \leqslant 0$ for $0 \leqslant \kappa \leqslant 1/2$ and the other term in the exponent is negative, $p^* \leqslant 2^{n(c\kappa/8)(1-\Phi)\log((1+\varepsilon)\kappa)}$. Finally, we use again $\log((1+\varepsilon)\kappa) \leqslant -0.9$ to obtain $p^* \leqslant 2^{-b \cdot s}$, where $b = 0.9c(1-\Phi)/8$ does not depend on κ. Since $s = \Omega(n)$, it holds that $p^* \leqslant 1/n^2$ for large enough n. ◀

▶ **Theorem 8.** *For any constant $\Phi \in (0, 1/10]$ and for any $O(\log n)$-regular expander graph $G = (V, E)$ with constant conductance, Procedure* ExpanderDegreeReduction *uses $O(\log^3 n)$ rounds and $O(n \log^3 n)$ messages to output an $O(1)$-bounded degree expander graph with conductance Φ with high probability.*

Proof. We start with the round complexity. By the algorithm definition, we run $O(\log n)$ phases, and within each phase, tokens take at most $\ell = O(\log n)$ steps. Moreover, a simple randomized analysis shows that each node receives, for any $0 \leqslant i \leqslant \ell$, in expectation $O(c) = O(1)$ tokens having taken i steps and per incident edge. Thus, by Chernoff bounds, each node receives, with high probability, at most $O(\log n)$ tokens having taken i steps, for any $0 \leqslant i \leqslant \ell$. (A more detailed analysis can be found in the proof of Lemma 3.2 in [18].) Thus, under the condition that tokens having taken less steps have priority to be sent, all tokens take ℓ steps within $O(\log^2 n)$ rounds with high probability. As for the message complexity, each node can send at most 1 messages per round, thus the message complexity follows from the round complexity. Next, the resulting graph trivially has bounded degree since by the algorithm definition, each node has up to $c + \delta \leqslant 2\delta$ incident edges.

It remains to show that the resulting graph has constant conductance with high probability. To do so, we give a similar proof to that of Lemma 1 in [11]. To start with, all nodes are satisfied when the algorithm ends, by Lemma 6. We consider an arbitrary $S \subset V$ of size $s \leqslant n/2$ and the following random variables: the edges $\mathcal{E}_1, \dots, \mathcal{E}_{cs}$ obtained via the algorithm, each corresponding to a now inactive token originating in S. After which, let the random variable $\mathcal{L}(S)$ denote the number of edges with both endpoints in S. We shall upper bound $\Pr[\mathcal{L}(S) \geqslant (1-\Phi)cs]$. For any integer $i \in \{1, \dots, cs\}$, let $\mathbb{1}_i$ be the indicator random variable indicating whether $\mathcal{E}_i$ has both endpoints in S. Since each token is obtained from a random walk of length at least ℓ, where ℓ is greater than the mixing time of G, then for any integer $i \in \{1, \dots, cs\}$, it holds by union bound that $\Pr[\mathbb{1}_i = 1] \leqslant (1+\varepsilon)s/n$. Conditioning on the events $\mathbb{1}_1 = 1, \dots, \mathbb{1}_{i-1} = 1$ can only reduce that probability, since nodes (in S) can hold a maximum of δ inactive tokens. Thus, $\Pr\Big[\mathbb{1}_i = 1 \mid \bigwedge_{j=1}^{i-1} \mathbb{1}_j = 1\Big] \leqslant (1+\varepsilon)s/n$. By the chain rule of conditional probability, we have $\Pr\Big[\bigwedge_{j=1}^{i} \mathbb{1}_j = 1\Big] = \Pr\Big[\mathbb{1}_i = 1 \mid \bigwedge_{j=1}^{i-1} \mathbb{1}_j = 1\Big] \cdot \Pr\Big[\bigwedge_{j=1}^{i-1} \mathbb{1}_j = 1\Big] = \Pr[\mathbb{1}_1 = 1] \cdot \prod_{j=2}^{cs} \Pr\Big[\mathbb{1}_j = 1 \mid \bigwedge_{k=1}^{j-1} \mathbb{1}_k = 1\Big] \leqslant ((1+\varepsilon)s/n)^i$. Note that in an analogous coin-flipping experiment with cs coins, the probability that you get at least $(1-\Phi)cs$ coins is upper bounded by the probability that you get $(1-\Phi)cs$ heads and leave other coins unobserved. Thus, $\Pr[\mathcal{L}(S) \geqslant (1-\Phi)cs] \leqslant \binom{cs}{(1-\Phi)cs}((1+\varepsilon)s/n)^{(1-\Phi)cs}$.

From the above inequality, we get $p^* = \Pr[\exists S, |S| = s \text{ and } \mathcal{L}(S) \geqslant (1-\Phi)cs] \leqslant \binom{n}{s}\binom{cs}{(1-\Phi)cs}((1+\varepsilon)s/n)^{(1-\Phi)cs}$. By Lemma 7, $p^* \leqslant 1/n^2$ for large enough n and some suitably chosen integer $c \geqslant 1$ and constant $\varepsilon \geqslant 0$ (where ε can be made as small as required by taking $\ell = O(\log n)$ large enough). It suffices to union bound over all $n/2$ possible sizes for S (for $s \in \{1, \dots, n/2\}$) to get that the resulting graph has constant conductance Φ with probability at least $1 - 1/n$. ◀

3.2 Computing Graph Sketches via Gossip

Consider a node subset $S \subset V$ and a set of (peer-to-peer) edges $E' \subseteq V^2$, such that the graph $G' = (S, E')$ is an $O(1)$-bounded degree expander graph (and thus connected), whose maximum degree d is known to all nodes (in S). We describe the Aggregate-Sketch-Vector primitive run on G'. We assume that all nodes in S know the minimum ID among the nodes in S, that all nodes in S have computed a common graph sketch matrix M_G, with certain properties (see Lemma 11), and that each node $u \in S$ has computed the corresponding sketch vector $\mathbf{s}_G(u)$. Primitive Aggregate-Sketch-Vector computes the aggregate sketch vector $\mathbf{s}_G(S) = \sum_{u \in S} \mathbf{s}_G(u)$.

Description. Each node $u \in S$ creates two sketch vectors, a positive sketch vector and a negative sketch vector, denoted by $\mathbf{s}_G^+(u)$ and $\mathbf{s}_G^-(u)$, respectively. $\mathbf{s}_G^+(u)$ holds all positive value entries of $\mathbf{s}_G(u)$ and the remaining entries are zero. $\mathbf{s}_G^-(u)$ is similarly defined. Define $\mathbf{s}_G^+(S) = \sum_{u \in S} \mathbf{s}_G^+(u)$ and $\mathbf{s}_G^-(S) = \sum_{u \in S} \mathbf{s}_G^-(u)$. All nodes u run two instances of Push-Sum (see [36]) for $T_s = O(\log n \log(nx))$ phases, where each phase consists of $T_p = O(\log^2 n)$ rounds and x comes from the graph sketching matrix M_G (see Lemma 11), to obtain $\mathbf{s}_G^+(S)$ and $\mathbf{s}_G^-(S)$. Note that Push-Sum, as described in [36], was described for a complete graph and is a gossip-style technique to compute aggregate functions (like average, sum, etc.) in a network. The description below shows how to simulate it on an $O(1)$-bounded degree expander to compute the sum.

We describe an instance of Push-Sum from the perspective of node u to obtain $\mathbf{s}_G^+(S)$. The process is similar to obtain $\mathbf{s}_G^-(S)$. At the end of each phase t, each node u maintains some weight value $w_{t,u}^+$ and some estimate of the average of the sketches $s_{t,u}^+$. After a sufficiently long time $t^* = T_s T_p + 1$ has passed, the ratio $s_{t^*,u}^+ / w_{t^*,u}^+$ will be an approximation of $\mathbf{s}_G^+(S)$.

Initially, the minimum ID node min sets its weight $w_{0,\min}^+ = 1$ and the remaining nodes u set their weights $w_{0,u}^+ = 0$. Initially, every node u sets $s_{0,u}^+ = \mathbf{s}_G^+(S)$. We assume that at the end of the phase 0 (i.e., before the algorithm begins), each node u sends the pair $(w_{0,u}^+, s_{0,u}^+)$ to itself.

In each phase $t \geqslant 1$ of Push-Sum, each node u does the following. Let $\{(\hat{s_r}, \hat{w_r})\}$ be all pairs sent to node u at the end of phase $t-1$. Node u computes $s_{t,u}^+ = \sum_r \hat{s_r}$ and $w_{t,u}^+ = \sum_r \hat{w_r}$.[14] Node u constructs the pair $(\frac{1}{2}s_{t,u}^+, \frac{1}{2}w_{t,u}^+)$ and sends it to itself and a node $v \in S$ chosen uniformly at random as follows. Node u initiates a lazy random walk of length $O(\log n)$ within G' carrying the message $(\frac{1}{2}s_{t,u}^+, \frac{1}{2}w_{t,u}^+)$. During these T_p rounds, node u helps other messages continue their random walks. After T_s phases are over, i.e, at time $t^* = T_s T_p + 1$ rounds, an approximation of $\mathbf{s}_G^+(S)$ is constructed at node u as $s_{t^*,u}^+ / w_{t^*,u}^+$. The correct values of $\mathbf{s}_G^+(S)$ can be recovered from $s_{t^*,u}^+ / w_{t^*,u}^+$ by rounding each element in $s_{t^*,u}^+ / w_{t^*,u}^+$ to the nearest legal value, i.e., the value that an element in the sketch vector can take.

Now, each node has computed $\mathbf{s}_G^+(S)$ and $\mathbf{s}_G^-(S)$, and computes the output $\mathbf{s}_G(S)$ as $\mathbf{s}_G(S) = \mathbf{s}_G^+(S) + \mathbf{s}_G^-(S)$.

Analysis. In order to capture the properties of Aggregate-Sketch-Vector, we must first look at the properties of Push-Sum. The following lemma from [36] captures the properties of Push-Sum. To bridge the notation, notice that each node u contains some initial sketch vector $x_u = \mathbf{s}_G^+(u)$ (and for another instance of Push-Sum $x_u = \mathbf{s}_G^-(u)$) and our goal is to calculate $\sum_{j \in S} x_j$.

▶ **Lemma 9** (Theorem 3.1 in [36]).

1. *With probability at least $1-\delta$, there is a time $t_0 = O(\log n + \log \frac{1}{\varepsilon} + \log \frac{1}{\delta})$, such that for all times $t \geqslant t_0$ and all nodes u, the relative error in the estimate of the average at node u is at most $\varepsilon \cdot \frac{\sum_j |x_j|}{|\sum_j x_j|}$ (where the relative error is $\frac{1}{|\sum_j x_j|} \cdot |\frac{s_{t,u}}{w_{t,u}} - \frac{1}{n} \cdot \sum_j x_j|$). In particular, the relative error is at most ε whenever all values x_j have the same sign.*
2. *The sizes of all messages sent at time t are bounded by $O(t + \max_j bits(x_j))$ bits, where $bits(x_j)$ denotes the number of bits in the binary representation of x_j.*

[14] Recall that $s_{t,u}^+$ is some vector. The sum of vectors denotes the sum of the elements for each index of the vectors.

As mentioned in [36], we can get the sum of the values of all nodes instead of just the average, by setting the initial weights such that only one node has weight 1 and the remaining have weight 0, as we do in Aggregate-Sketch-Vector.

It should be noted that the original Push-Sum was designed for a complete graph, where each node u could sample one of the nodes of the graph uniformly at random. We simulate this process on an $O(1)$-bounded degree expander by having node u choose another node as the result of a lazy random walk that is run for the mixing time. This guarantees us that we may sample a node in S nearly uniformly at random.

Since this requires multiple nodes initiating and facilitating random walks simultaneously, we make use of the following lemma, adapted from a lemma in [18]. We note that it is for the traditional CONGEST model, where every node can communicate with all of its neighbors in a given round, unlike the current setting. We explain in the analysis of the final lemma of this section, how the following lemma can easily be adapted to the current model.

▶ **Lemma 10** (Adapted from Lemma 3.2 in [18]). *In the traditional CONGEST model, let $G = (V, E)$ be an undirected graph and let each node $v \in V$, with degree $d(v)$, initiate $\eta d(v)$ random walks, each of length λ. Then all walks reach their destinations in $O(\eta\lambda \log n)$ rounds with high probability.*

The following lemma shows that Aggregate-Sketch-Vector works as desired to help us reconstruct the aggregate of the initial sketch vectors in the desired time.

▶ **Lemma 11.** *Assume that each node u has a graph sketch vector $\boldsymbol{s}_G(u)$ computed using a graph sketching matrix M_G that satisfies the following properties:*

- *For all nodes u, elements in $\boldsymbol{s}_G(u)$ belong to the same range of values $[L, U]$, where $L, U \in \mathbb{R}$, and $U - L = x \neq 0$.*
- *The range of values taken by elements in the sketch vector, Range, is some totally ordered countable set of numbers such that the minimum distance between any two numbers is at least some constant $c > 0$, i.e., $\forall u, v \in Range$ if $u \neq v$, then $|u - v| \geqslant c$.*

If all nodes $u \in S$ participate in Aggregate-Sketch-Vector for $O(\log^3 n \log(nx))$ rounds, then each node outputs the aggregated sketch vector $\boldsymbol{s}_G(S) = \sum_{u \in S} \boldsymbol{s}_G(u)$ with high probability.

Proof. We first argue that Push-Sum is faithfully simulated. Each round of the original Push-Sum corresponds to $O(\log n)$ phases of the process as described here. We first describe the end result of one phase. In one phase, we ensure that a lazy random walk starting at some node u is run for enough rounds to reach mixing time. In an $O(1)$-bounded degree expander, the mixing time of one single lazy random walk is $O(\log n)$ rounds. However, since each node simultaneously initiates a single lazy random walk of length $O(\log n)$, by Lemma 10, we see that we need $O(\log^2 n)$ rounds to ensure that they all complete. Furthermore, it should be noted that Lemma 10 applies to the traditional CONGEST model. However, in a bounded degree graph with maximum degree d, one round of the traditional CONGEST model can be simulated in d rounds in the current model. Since $d = O(1)$, we incur no overhead with respect to the lemma and see that the $O(1)$ random walks initiated from each node, each run for $O(\log n)$ time, require in total $O(\log^2 n)$ rounds to complete. Thus in one phase, some node from G' will be sampled (nearly) uniformly at random.

To faithfully simulate Push-Sum, we want the sampled node to not be the starting node. However, in order to ensure that some node, other than the starting node, is sampled uniformly at random, we need to run $O(\log n)$ phases of this protocol. By a simple Chernoff bound, we can see that with high probability, even when $|S| = O(1)$, a random walk starting at some node u will sample some node that is not u.

Notice that by separating each node's sketch vector into two vectors corresponding to positive values and negative values and using PUSH-SUM to find the aggregate of each set of vectors separately, we ensure that the signs of values being aggregated is the same. From Lemma 9, we see that the relative error of any entry is thus at most ε. By setting $\varepsilon = 1/nx$, we see that the output of each element of the sketch vector is within $1/n$ of the actual value. Since each element of the sketch vector can only take values such that the distance between values is some constant c, it is easy to see that rounding the element to the nearest legal value gives the correct value (since the outputted value will only be at most some $1/n \ll c/2$ from a legal value).

Finally, in order to ensure that these guarantees are with high probability, it is sufficient to set $\delta = 1/n$ in Lemma 9. Substituting the values of ε and δ in Lemma 9, we get the corresponding run time. ◀

In our main algorithm in Section 4, AGGREGATE-SKETCH-VECTOR is used to compute the aggregate of the sketch vectors that are obtained via the graph sketching matrix guaranteed by Lemma 1 in Section 2.2. That matrix satisfies the requirements of Lemma 11 and $x = O(n^r)$, for some positive constant $r \geqslant 1$, and $c = 1$. Thus, we have the following corollary.

▶ **Corollary 12.** *Consider any node subset $S \subset V$ and set of (possibly peer-to-peer) edges $E' \subseteq V^2$, such that the graph $G' = (S, E')$ is an $O(1)$-bounded degree expander graph. Assume that each node u has a graph sketch vector $\boldsymbol{s}_G(u)$ computed using the graph sketching matrix guaranteed by Lemma 1 in Section 2.2. If all nodes $u \in S$ participate in AGGREGATE-SKETCH-VECTOR for $O(\log^4 n)$ rounds, then each node will obtain the aggregated sketch vector $\boldsymbol{s}_G(S) = \sum_{u \in S} \boldsymbol{s}_G(u)$ with high probability.*

4 Overlay Construction Protocol

Let $G = (V, E)$ be the original graph and let $\Phi \in (0, 1/10]$ be the desired conductance. We show how to transform any arbitrary graph $G = (V, E)$ (even with large degree) into an expander overlay network with conductance $\geqslant \Phi$ and bounded degree. We assume the harder case of $\Phi = \Omega(1)$, or in other words, of building an overlay network with conductance $\Omega(1)$.

Initially, each node forms its own (high conductance) cluster, and the set of intra-cluster edges E_0 is empty. In each stage $i \in [1, k]$, we compute a new set of edges $E_i \subseteq V^2$ that merge the stage's starting clusters into fewer and larger-sized (high conductance) clusters. In fact, the merging reduces the number of clusters by half with constant probability. After $k = O(\log n)$ stages, the resulting edge set E_k generates an expander graph $G(E_k) = (V, E_k)$ with high conductance (i.e., at least Φ).

4.1 Algorithm Description

The algorithm runs in $k = O(\log n)$ stages, each consisting of three steps. We ensure the following invariant holds at the start of each stage $i \in [1, k]$: the graph generated by E_{i-1}, $G(E_{i-1}) = (V, E_{i-1})$, is decomposed into clusters (i.e., connected components) with constant conductance Φ and constant maximum degree. (Note that the initial edge set E_0 is empty and thus the invariant holds trivially.)

First Step. Nodes spread, within each cluster of $G(E_{i-1})$, the minimum ID and an associated $O(\log^2 n)$ random bit string, by executing a PUSH style information spreading algorithm (e.g., see [21] or [25]) over (each cluster of) $G(E_{i-1})$ for $T_g = O(\Phi^{-1} \log n)$ rounds. More concretely, nodes do the following. Initially, each node picks an $O(\log^2 n)$ bit random string. Then, in each round, each node chooses one random neighbor in $G(E_{i-1})$ and sends a message containing $\langle$minimum ID seen so far, associated random string$\rangle$ to it. Since $G(E_{i-1})$ decomposes into clusters of maximum degree $O(1)$ and of conductance $\geqslant \Phi$, and thus of diameter $O(\Phi^{-1} \log n)$, the result of [21] implies that T_g rounds is sufficient to spread, within each cluster, that cluster's minimum ID and its associated $O(\log^2 n)$ bit random string.

Second Step. Now, all nodes within some cluster V_j of $G(E_{i-1})$ know the minimum ID of that cluster and the associated $O(\log^2 n)$ bit random string. Then, nodes use this shared randomness to sample one edge per cluster, using graph sketches. We describe how this is done in the next paragraph. Each such sampled edge is an inter-cluster edge (i.e., its endpoints are in different clusters) with constant probability. These sampled inter-cluster edges – the set of which is denoted by E_i^c – allow to merge clusters of $G(E_{i-1})$, reducing them by a constant fraction with constant probability. However, although each cluster samples a single edge, each node may have $\omega(1)$ incident edges in E_i^c: for example, if many clusters sample edges incident to one particular node. Thus, we finish the step by computing a second edge set E_i^b such that $G(E_i^b)$ has bounded degree and preserves the connectivity of $G(E_i^c)$.

Let v be any node within some cluster V_j of $G(E_{i-1})$. First, each node v computes its sketch vector $\mathbf{s}_G(v)$ (using the cluster's shared random string to generate a graph sketching matrix, see Subsection 2.2) and runs Procedure Aggregate-Sketch-Vector. Its output is the aggregated sketch vector $\sigma(v) = \sum_{v \in V_j} \mathbf{s}_G(v)$. Next, v samples an edge using $\sigma(v)$ (see Subsection 2.2), and this edge is an inter-cluster edge with some constant probability δ. If v is an endpoint of that edge, then it contacts the other endpoint node, they exchange information on their cluster's minimum ID and v drops the edge if these IDs are identical. In other words, all sampled edges whose endpoints come from different clusters (i.e., exchanged different IDs) are added to the edge set E_i^c, whereas others are simply "dropped".

Now, it remains to compute E_i^b, by applying a simple degree reduction procedure on E_i^c. Note that for each edge in E_i^c, one node belongs to the cluster that sampled that edge. That node is said to own the edge and one can, for the sake of the procedure's description, orient the edge from the owner node to the other endpoint. Then, nodes do the following in two rounds. Initially, nodes add all incident edges in E_i^c to E_i^b. Any node u with at least 3 incoming incident edges in E_i^b – the set of their owners is denoted by N_u – locally computes an arbitrary undirected cycle over $N_u \cup \{u\}$, and locally replaces these incident edges in E_i^b with its two incident edges in that cycle. Then, each node that owns an edge contacts the other endpoint, and they exchange the result of their local computations (if any). Finally, nodes that received a cycle locally replace their owned edge in E_i^b with their two incident edges in that cycle.

Third Step. Finally, we transform each cluster of $G(E_{i-1} \cup E_i^b)$ into an $O(1)$-bounded degree expander graph with constant conductance Φ. More concretely, we compute a set of edges E_i for which: (i) each cluster of $G(E_i)$ is a cluster of $G(E_{i-1} \cup E_i^b)$, and (ii) each cluster of $G(E_i)$ is an $O(1)$-bounded degree expander graph with constant conductance Φ.

To do so, we first run Procedure CreateExpander (described in Subsection 2.3) for $O(\log^2 n)$ rounds. This computes, for each cluster of $G(E_{i-1} \cup E_i^b)$, an $O(\log n)$-regular expander graph with constant conductance Φ. Next, we run Procedure ExpanderDegreeReduction (described in Subsection 3.1) for $O(\log^3 n)$ rounds to reduce the degree of these expander graphs to some constant. More precisely, this procedure computes, for each $O(\log n)$-regular expander graph, an $O(1)$-bounded degree expander graph that also has constant conductance Φ. Adding together the edges of all these $O(1)$-bounded degree expander graphs gives the edge set E_i.

4.2 Analysis

To prove the correctness of our overlay construction algorithm and bound its round and message complexities (see Theorem 17), we first give a series of lemmas that rely upon the following invariant: for any stage $i \in [1, k]$, all clusters of $G(E_{i-1})$ are $O(1)$-bounded degree expander graphs with constant conductance Φ. We start by showing that each edge sampled using graph sketching is an inter-cluster edge with constant probability.

▶ **Lemma 13.** *For any stage $i \in [1, k]$, assume all clusters of $G(E_{i-1})$ are $O(1)$-bounded degree expander graphs with constant conductance Φ. Then, each cluster of $G(E_{i-1})$ samples an inter-cluster edge with probability at least $3/4$.*

Proof. Since we use the information spreading algorithm of [25] (see Theorem 1 in [25]), and all clusters of $G(E_{i-1})$ are $O(1)$-bounded degree expander graphs with constant conductance Φ, then for each such cluster, the minimum ID and the associated $O(\log^2 n)$ bits random string is spread to all of that cluster's nodes in $O(\Phi^{-1} \log n)$ rounds w.h.p.

Next, consider any cluster V_j of $G(E_{i-1})$. Recall that each node $v \in V_j$ computes its sketch vector $\mathbf{s}_G(v)$ initially and uses it as input for Procedure Aggregate-Sketch-Vector. Its output is $\sigma(v) = \sum_{u \in V_j} \mathbf{s}_G(u)$, the aggregate of sketch vector within cluster V_j, with high probability by Corollary 12. Then, by choosing $\delta = 1/4$, each node can sample an inter-cluster edge from this aggregate vector with constant probability $1 - \delta = 3/4$, by Lemma 1. ◀

As a result of the above lemma, many sampled edges are inter-cluster with constant probability. Thus, the addition of these sampled edges significantly reduces the number of clusters – in fact, by a constant fraction – with constant probability.

▶ **Lemma 14.** *For any stage $i \in [1, k]$, assume all $c \geqslant 1$ clusters of $G(E_{i-1})$ are $O(1)$-bounded degree expander graphs with constant conductance Φ. If $G(E_{i-1})$ has $c > 1$ clusters, then $G(E_{i-1} \cup E_i^c)$ has at most $3c/4$ clusters with probability at least 1/2.*

Proof. All clusters of $G(E_{i-1})$ are $O(1)$-bounded degree expander graphs with constant conductance Φ (from the lemma's assumption). Then, by Lemma 13, each cluster of $G(E_{i-1})$ samples an inter-cluster edge with probability $3/4$. By linearity of expectation, in expectation $3c/4$ of the sampled edges are inter-cluster edges, or equivalently, in expectation $c/4$ sampled edges are intra-cluster edges. Applying Markov's inequality, the probability that more than $c/2$ sampled edges are intra-cluster is at most $1/2$, or equivalently, the probability that at least $c/2$ sampled edges are inter-cluster is at least $1/2$. Any of these inter-cluster edges (say, from V_j to $V_{j'}$) allows to merge two clusters and reduce the number of clusters by 1, unless an inter-cluster edge from $V_{j'}$ to V_j was also sampled. Hence, if there are at least a inter-cluster edges, then $G(E_{i-1} \cup E_i^c)$ has at most $c - a/2$ clusters. Thus, $G(E_{i-1} \cup E_i^c)$ has at most $3c/4$ clusters with probability at least $1/2$. ◀

Note that the sampled inter-cluster edges may generate a graph with large degree. Next, we prove that the degree reduction procedure used in the second step is correct.

▶ **Lemma 15.** *For any stage $i \in [1, k]$, $G(E_i^b)$ has maximum degree at most 4 and preserves the connectivity of $G(E_i^c)$, that is, for any edge $(u, v) \in E_i^c$, u and v are connected in $G(E_i^b)$.*

Proof. Consider an arbitrary node u. As in the algorithm description, for each edge of E_i^c, assume for the sake of the proof that it is directed from the sampling cluster outwards. (Note that E_i^c contains no edges with both endpoints in the same cluster.) We show that u is incident to at most 2 edges in E_i^b due to incoming edges in E_i^c, and to at most 2 other edges in E_i^b due to outgoing edges in E_i^c; thus, u has a degree of at most 4 in $G(E_i^b)$. On the one hand, if u has at least 3 incoming edges in E_i^c, then these edges are replaced by only 2 edges (from the cycle built by u in the second half of the second step) in E_i^b. On the other hand, each cluster, and thus node, is incident to at most one outgoing edge in E_i^c (by the algorithm definition). This outgoing edge may be replaced by at most two edges in E_i^b (when the outgoing edge's other endpoint has more than 3 incoming edges in E_i^c).

Finally, it is straightforward to show that $G(E_i^b)$ preserves the connectivity of $G(E_i^c)$. Indeed, any edge $(u, v) \in E_i^c$ that does not remain in E_i^b is replaced by a cycle in E_i^b (locally computed by either u or v) that includes both u and v. ◀

Next, we show that the invariant is maintained, and that the stage reduces the number of clusters by a constant fraction with constant probability.

▶ **Lemma 16.** *For any stage $i \in [1, k]$, assume all $c \geqslant 1$ clusters of $G(E_{i-1})$ are $O(1)$-bounded degree expander graphs with constant conductance Φ. Then, all $c' \leqslant c$ clusters of $G(E_i)$ are $O(1)$-bounded degree expander graphs with constant conductance Φ. Moreover, if $G(E_{i-1})$ has $c > 1$ clusters, then $G(E_i)$ has at most $3c/4$ clusters with probability 1/2.*

Proof. Consider some stage $i \in [1, k]$. Let the $c \geqslant 1$ clusters of $G(E_{i-1})$ be denoted by $V_1, \ldots, V_c$. We first provide some properties about $G(E_{i-1} \cup E_i^b)$. To start with, $G(E_{i-1})$ has constant maximum degree (from the lemma's assumption) and thus by Lemma 15, $G(E_{i-1} \cup E_i^b)$ also has constant maximum degree, denoted by d. Second, each cluster V_i of $G(E_{i-1})$ is part of (i.e., a subset of) some cluster in $G(E_{i-1} \cup E_i^b)$, as the latter graph only has additional edges. Third, if $G(E_{i-1})$ has $c > 1$ clusters, then $G(E_{i-1} \cup E_i^b)$ has at most $c/2$ clusters with constant probability. Indeed, $G(E_{i-1} \cup E_i^c)$ has at most $3c/4$ clusters with probability at least 1/2. Since $G(E_{i-1} \cup E_i^b)$ preserves the connectivity of $G(E_{i-1} \cup E_i^c)$ by Lemma 15, $G(E_{i-1} \cup E_i^b)$ also has at most $3c/4$ clusters with probability at least 1/2.

Finally, recall that the edge set E_i is obtained by nodes running Procedure CreateExpander (with parameters Φ and d) followed by Procedure ExpanderDegreeReduction on each cluster of $G(E_{i-1} \cup E_i^b)$. First, note that Procedures CreateExpander and ExpanderDegreeReduction do not disconnect any clusters of $G(E_{i-1} \cup E_i^b)$, and thus in particular $G(E_i)$ has at most as many clusters as $G(E_{i-1} \cup E_i^b)$. Moreover, by Lemma 2 and Theorem 8, each cluster of $G(E_i)$ is an $O(1)$-bounded degree expander graph with constant conductance Φ. This completes the proof. ◀

Note that by definition, any graph has at least one cluster. The above lemma implies that within $O(\log n)$ stages, we obtain a graph with exactly one high-conductance cluster, and thus solve the overlay construction problem – see the following theorem.

▶ **Theorem 17.** *The overlay construction problem can be solved with high probability in $O(\log^5 n)$ rounds and $\tilde{O}(n)$ messages.*

Proof. To start with, for any given stage $i \in [1, k]$, the stage is said to be *successful* if $G(E_i)$ either (i) has a single cluster or (ii) has less than 3/4 as many clusters as $G(E_{i-1})$. By Lemma 16 (and a simple induction on i), each stage is successful with probability at least 1/2. Hence, for a large enough number $k = O(\log n)$ of stages, a simple application of Chernoff bounds imply that there are at least $\log_{4/3} n$ successful stages. Thus, $G(E_k)$ has a single cluster. Moreover, by Lemma 16 again, $G(E_k)$ is an $O(1)$-bounded degree expander graph with constant conductance Φ. The correctness follows.

The round complexity of $O(\log^5 n)$ is straightforward: $O(\log n)$ stages each take $O(\log^4 n)$ rounds. The message complexity follows directly from the time complexity, the fact that communication is always done on graphs of degree at most $O(\log n)$, and that messages of size $O(\text{polylog}\, n)$ suffice throughout the algorithm (both to share the random string in the first step and by Lemma 9, during Procedure Push-Sum in the second step). ◀

5 Experimental Results

The proposed overlay construction protocol is implemented in a sequential simulation to study properties of the algorithm for a few different types of low-conductance graphs. We study the number of rounds and the conductance of the resulting graphs. The simulation follows the algorithm's steps with some small deviations.

The algorithm is implemented in sequential simulated form using Python and the graph library graph_tool [47], to study properties of the algorithm for a few low-conductance graphs with different properties, and study the number of rounds and estimate the conductance provided by the algorithm. A full discussion of the implementation is provided in the full version. The types of graphs tested by the simulator include a high-diameter cycle graph *circle-10000*, a graph on a square grid *grid-50-50*, a randomly-generated preferential attachment graph *barabasi-2000-2-2*, as well as modestly sized real-world graphs with differing topologies: graphs modeling disease contagion *kissler*, social network attachment *twitch*, and citation networks *wiki*. Table 1 summarizes the results.

Table 1 Table showing simulation results on various input graphs (denoted by G) and the corresponding graphs output by the protocol (denoted by G_E). n is the number of nodes of G. Phases denotes the number of phases of the overlay construction protocol that were required to produce G_E. D_G and D_{G_E} are lower-bound estimates of the graph diameter of G and G_E. Φ_G and Φ_{G_E} are upper-bound estimates of conductance of G and G_E.

Graph	n	Phases	D_G	D_{G_E}	Φ_G	Φ_{G_E}
circle-10000	10000	6	1111	6	0.068	0.453
grid-50-50	2500	5	98	4	0.148	0.449
barabasi-2000-2-2	2000	3	5	4	0.4	0.451
wiki	2277	3	16	4	0.08	0.450
twitch	7126	3	10	5	0.143	0.452
kissler	409	3	9	3	0.2	0.446

The conductance of the input and the final overlay G and G_E are each estimated using $O(n)$ sampled graph cuts, to provide an upper-bound estimate on the actual graph conductances Φ_G and Φ_{G_E}. The table shows for each input graph these conductance estimates as well as the number of phases of the protocol required for the algorithm to terminate. The table also gives the estimate of the diameters D_G and D_{G_E} of the initial and final graphs respectively, given by the pseudo-diameter as calculated for a sampling of nodes. The results show that the conductance of the constructed graph is likely significantly higher compared to the starting graph and is close to 0.5 which is essentially the best possible value for a constant-degree random graph. The results also show that the diameter of the final expander is roughly in line with expectations of an $O(\log n)$ bound, and that the number of rounds, conductance, and diameter of G_E are independent of the edge density of the initial graph.

6 Conclusion

In this paper, we presented the first distributed overlay construction protocol that is fast (taking $O(\log^5 n)$ rounds) as well as taking significantly less communication (using $\tilde{O}(n)$ messages, regardless of the number of edges of the initial graph). The protocol assumes the P2P-GOSSIP model which uses gossip-based communication (which is very lightweight) and the reconfigurable nature of P2P networks. Both bounds are essentially the best possible. Our result also implies that the distributed complexity of solving fundamental problems such as broadcast, leader election, and MST construction is significantly smaller in the P2P-GOSSIP model compared to the standard CONGEST model.

Several open questions remain. One is to improve the round complexity of our protocol. In particular, can we improve the round complexity to $O(\log^2 n)$ rounds while keeping $\tilde{O}(n)$ communication? Another interesting follow up work is to adapt our protocol to work under a churn setting. A third interesting research direction is to investigate the complexity of other fundamental problems such as computing shortest paths in the P2P-GOSSIP model.

References

1 Kook Jin Ahn, Sudipto Guha, and Andrew McGregor. Analyzing graph structure via linear measurements. In *Proceedings of the 23rd Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 459–467, 2012.

2 Kook Jin Ahn, Sudipto Guha, and Andrew McGregor. Graph sketches: sparsification, spanners, and subgraphs. In *Proceedings of the 31st ACM Symposium on Principles of Database Systems (PODS)*, pages 5–14, 2012.

3 Dana Angluin, James Aspnes, Jiang Chen, Yinghua Wu, and Yitong Yin. Fast construction of overlay networks. In *Proceedings of the Seventeenth Annual ACM Symposium on Parallelism in Algorithms and Architectures*, pages 145–154, 2005.

4 Dana Angluin, James Aspnes, Jiang Chen, Yinghua Wu, and Yitong Yin. Fast construction of overlay networks. In Phillip B. Gibbons and Paul G. Spirakis, editors, *SPAA 2005: Proceedings of the 17th Annual ACM Symposium on Parallelism in Algorithms and Architectures, July 18-20, 2005, Las Vegas, Nevada, USA*, pages 145–154. ACM, 2005.

5 John Augustine, Soumyottam Chatterjee, and Gopal Pandurangan. A fully-distributed scalable peer-to-peer protocol for byzantine-resilient distributed hash tables. In Kunal Agrawal and I-Ting Angelina Lee, editors, *SPAA '22: 34th ACM Symposium on Parallelism in Algorithms and Architectures, Philadelphia, PA, USA, July 11 - 14, 2022*, pages 87–98. ACM, 2022. doi:10.1145/3490148.3538588.

6 John Augustine, Mohsen Ghaffari, Robert Gmyr, Kristian Hinnenthal, Christian Scheideler, Fabian Kuhn, and Jason Li. Distributed computation in node-capacitated networks. In Christian Scheideler and Petra Berenbrink, editors, *The 31st ACM on Symposium on Parallelism in Algorithms and Architectures, SPAA 2019, Phoenix, AZ, USA, June 22-24, 2019*, pages 69–79. ACM, 2019. doi:10.1145/3323165.3323195.

7 John Augustine, Anisur Rahaman Molla, Ehab Morsy, Gopal Pandurangan, Peter Robinson, and Eli Upfal. Storage and search in dynamic peer-to-peer networks. In *Proceedings of the Twenty-fifth Annual ACM Symposium on Parallelism in Algorithms and Architectures (SPAA)*, pages 53–62, 2013.

8 John Augustine, Gopal Pandurangan, and Peter Robinson. Fast byzantine agreement in dynamic networks. In *Proceedings of the ACM Symposium on Principles of Distributed Computing (PODC)*, pages 74–83, 2013.

9 John Augustine, Gopal Pandurangan, and Peter Robinson. Fast byzantine leader election in dynamic networks. In *29th International Symposium on Distributed Computing (DISC)*, volume 9363 of *Lecture Notes in Computer Science*, pages 276–291, 2015.

10 John Augustine, Gopal Pandurangan, and Peter Robinson. Distributed algorithmic foundations of dynamic networks. *SIGACT News*, 47(1):69–98, 2016.

11 John Augustine, Gopal Pandurangan, Peter Robinson, Scott Roche, and Eli Upfal. Enabling robust and efficient distributed computation in dynamic peer-to-peer networks. In *IEEE 56th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 350–369, 2015.

12 John Augustine, Gopal Pandurangan, Peter Robinson, and Eli Upfal. Towards robust and efficient computation in dynamic peer-to-peer networks. In *Proceedings of the Twenty-third Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 551–569, 2012.

13 Amitabha Bagchi, Ankur Bhargava, Amitabh Chaudhary, David Eppstein, and Christian Scheideler. The effect of faults on network expansion. *Theory Comput. Syst.*, 39(6):903–928, 2006.

14 Keren Censor-Hillel, Bernhard Haeupler, Jonathan A. Kelner, and Petar Maymounkov. Global computation in a poorly connected world: fast rumor spreading with no dependence on conductance. In Howard J. Karloff and Toniann Pitassi, editors, *Proceedings of the 44th Symposium on Theory of Computing Conference, STOC 2012, New York, NY, USA, May 19 - 22, 2012*, pages 961–970. ACM, 2012. doi:10.1145/2213977.2214064.

15 Keren Censor-Hillel and Hadas Shachnai. Fast information spreading in graphs with large weak conductance. *SIAM J. Comput.*, 41(6):1451–1465, 2012. doi:10.1137/110845380.

16 Soumyottam Chatterjee, Gopal Pandurangan, and Nguyen Dinh Pham. Distributed mst: A smoothed analysis. In *Proceedings of the 21st International Conference on Distributed Computing and Networking*, ICDCN '20, New York, NY, USA, 2020. Association for Computing Machinery. doi:10.1145/3369740.3369778.

17 Colin Cooper, Martin E. Dyer, and Catherine S. Greenhill. Sampling regular graphs and a peer-to-peer network. In *Proceedings of the Sixteenth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 980–988. SIAM, 2005.

18 Atish Das Sarma, Danupon Nanongkai, Gopal Pandurangan, and Prasad Tetali. Distributed random walks. *J. ACM*, 60(1), 2013.

19 Alan J. Demers, Daniel H. Greene, Carl Hauser, Wes Irish, John Larson, Scott Shenker, Howard E. Sturgis, Daniel C. Swinehart, and Douglas B. Terry. Epidemic algorithms for replicated database maintenance. In Fred B. Schneider, editor, *Proceedings of the Sixth Annual ACM Symposium on Principles of Distributed Computing, Vancouver, British Columbia, Canada, August 10-12, 1987*, pages 1–12. ACM, 1987.

20 Michael Dinitz, Michael Schapira, and Asaf Valadarsky. Explicit expanding expanders. *Algorithmica*, 78(4):1225–1245, 2017. doi:10.1007/s00453-016-0269-x.

21 Uriel Feige, David Peleg, Prabhakar Raghavan, and Eli Upfal. Randomized broadcast in networks. *Random Structures & Algorithms*, 1(4):447–460, 1990.

22 Robert G. Gallager, Pierre A. Humblet, and Philip M. Spira. A distributed algorithm for minimum-weight spanning trees. *ACM Transactions on Programming Languages and systems (TOPLAS)*, 5(1):66–77, 1983.

23 Mohsen Ghaffari, Fabian Kuhn, and Hsin-Hao Su. Distributed MST and routing in almost mixing time. In *Proceedings of the 2017 ACM Symposium on Principles of Distributed Computing (PODC)*, pages 131–140, 2017.

24 Mohsen Ghaffari and Jason Li. New distributed algorithms in almost mixing time via transformations from parallel algorithms. In Ulrich Schmid and Josef Widder, editors, *32nd International Symposium on Distributed Computing, DISC 2018, New Orleans, LA, USA, October 15-19, 2018*, volume 121 of *LIPIcs*, pages 31:1–31:16. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2018. doi:10.4230/LIPIcs.DISC.2018.31.

25 George Giakkoupis. Tight bounds for rumor spreading in graphs of a given conductance. In *28th International Symposium on Theoretical Aspects of Computer Science (STACS 2011)*, pages 57–68, 2011.

26 Seth Gilbert, Gopal Pandurangan, Peter Robinson, and Amitabh Trehan. Dconstructor: Efficient and robust network construction with polylogarithmic overhead. In *Proceedings of the 39th Symposium on Principles of Distributed Computing*, pages 438–447, 2020.

27 C. Gkantsidis, M. Mihail, and A. Saberi. Random walks in peer-to-peer networks: Algorithms and evaluation. *Performance Evaluation*, 63(3):241–263, 2006.

28 Robert Gmyr, Kristian Hinnenthal, Christian Scheideler, and Christian Sohler. Distributed monitoring of network properties: The power of hybrid networks. In Ioannis Chatzigiannakis, Piotr Indyk, Fabian Kuhn, and Anca Muscholl, editors, *44th International Colloquium on Automata, Languages, and Programming, ICALP 2017, July 10-14, 2017, Warsaw, Poland*, volume 80 of *LIPIcs*, pages 137:1–137:15. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2017. doi:10.4230/LIPIcs.ICALP.2017.137.

29 Robert Gmyr and Gopal Pandurangan. Time-Message Trade-Offs in Distributed Algorithms. In *32nd International Symposium on Distributed Computing (DISC 2018)*, pages 32:1–32:18, 2018.

30 Thorsten Götte, Kristian Hinnenthal, Christian Scheideler, and Julian Werthmann. Time-optimal construction of overlay networks. In *Proceedings of the 2021 ACM Symposium on Principles of Distributed Computing (PODC)*, pages 457–468, 2021.

31 Shlomo Hoory, Nathan Linial, and Avi Wigderson. Expander graphs and their applications. *Bulletin of the American Mathematical Society*, 43(4):439–561, 2006.

32 Tim Jacobs and Gopal Pandurangan. Stochastic analysis of a churn-tolerant structured peer-to-peer scheme. *Peer-to-Peer Networking and Applications*, 6(1):1–14, 2013.

33 Hossein Jowhari, Mert Saglam, and Gábor Tardos. Tight bounds for lp samplers, finding duplicates in streams, and related problems. In Maurizio Lenzerini and Thomas Schwentick, editors, *Proceedings of the 30th ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems, PODS 2011, June 12-16, 2011, Athens, Greece*, pages 49–58. ACM, 2011. doi:10.1145/1989284.1989289.

34 Richard M. Karp, Christian Schindelhauer, Scott Shenker, and Berthold Vöcking. Randomized rumor spreading. In *41st Annual Symposium on Foundations of Computer Science, FOCS 2000, 12-14 November 2000, Redondo Beach, California, USA*, pages 565–574. IEEE Computer Society, 2000. doi:10.1109/SFCS.2000.892324.

35 D. Kempe and J. Kleinberg. Protocols and impossibility results for gossip-based communication mechanisms. In *Proceedings of The 43rd Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, November 2002.

36 David Kempe, Alin Dobra, and Johannes Gehrke. Gossip-based computation of aggregate information. In *44th Annual IEEE Symposium on Foundations of Computer Science, 2003. Proceedings.*, pages 482–491. IEEE, 2003.

37 Valerie King, Shay Kutten, and Mikkel Thorup. Construction and impromptu repair of an MST in a distributed network with o(m) communication. In Chryssis Georgiou and Paul G. Spirakis, editors, *Proceedings of the 2015 ACM Symposium on Principles of Distributed Computing, PODC 2015, Donostia-San Sebastián, Spain, July 21 - 23, 2015*, pages 71–80. ACM, 2015. doi:10.1145/2767386.2767405.

38 Shay Kutten, Gopal Pandurangan, David Peleg, Peter Robinson, and Amitabh Trehan. On the complexity of universal leader election. *Journal of the ACM*, 62(1):7:1–7:27, 2015. Invited paper from *ACM PODC* 2013. doi:10.1145/2699440.

39 C. Law and K.-Y. Siu. Distributed construction of random expander networks. In *IEEE INFOCOM*, pages 2133–2143, 2003.

40 Peter Mahlmann and Christian Schindelhauer. Distributed random digraph transformations for peer-to-peer networks. In *Proceedings of the Eighteenth Annual ACM Symposium on Parallelism in Algorithms and Architectures (SPAA)*, pages 308–317, 2006.

41 Yifan Mao, Soubhik Deb, Shaileshh Bojja Venkatakrishnan, Sreeram Kannan, and Kannan Srinivasan. Perigee: Efficient peer-to-peer network design for blockchains. In *ACM Symposium on Principles of Distributed Computing (PODC)*, pages 428–437, 2020.

42 Ali Mashreghi and Valerie King. Time-communication trade-offs for minimum spanning tree construction. In *Proceedings of the 18th International Conference on Distributed Computing and Networking (ICDCN)*, 2017.

43 Michael Mitzenmacher and Eli Upfal. *Probability and Computing: Randomized Algorithms and Probabilistic Analysis.* Cambridge University Press, 2^{nd} edition, 2017.

44 Gopal Pandurangan, Prabhakar Raghavan, and Eli Upfal. Building low-diameter P2P networks. In *IEEE Symposium on the Foundations of Computer Science (FOCS)*, pages 492–499, 2001.

45 Gopal Pandurangan, Peter Robinson, and Michele Scquizzato. Fast distributed algorithms for connectivity and mst in large graphs. *ACM Transactions on Parallel Computing (TOPC)*, 5(1):1–22, 2018.

46 Gopal Pandurangan and Amitabh Trehan. Xheal: a localized self-healing algorithm using expanders. *Distributed Computing*, 27(1):39–54, 2014.

47 Tiago P. Peixoto. The graph-tool python library. *figshare*, 2014. doi:10.6084/m9.figshare.1164194.

48 David Peleg. *Distributed Computing: A Locality-sensitive Approach.* Society for Industrial and Applied Mathematics, 2000.

49 Leslie G. Valiant. A scheme for fast parallel communication. *SIAM J. Comput.*, 11(2):350–361, 1982.