Enabling Robust and Efficient Distributed Computation in Dynamic Peer-to-Peer Networks

John Augustine*, Gopal Pandurangan†, Peter Robinson‡, Scott Roche§, and Eli Upfal¶

*Department of Computer Science & Engineering, Indian Institute of Technology Madras, Chennai, TN, 600036, India. E-mail: augustine@cse.iitm.ac.in.
†Department of Computer Science, University of Houston, Houston, TX 77204, USA. E-mail: gopalpandurangan@gmail.com.
‡Queen's University Belfast, United Kingdom. E-mail: p.robinson@qub.ac.uk.
§College of Computer and Information Science, Northeastern University, Boston MA 02115, USA. E-mail: str@ccs.neu.edu.
¶Department of Computer Science, Brown University, Providence, RI 02912, USA. E-mail: eli@cs.brown.edu.

Abstract

Motivated by the need for designing efficient and robust fully-distributed computation in highly dynamic networks such as Peer-to-Peer (P2P) networks, we study distributed protocols for constructing and maintaining dynamic network topologies with good expansion properties. Our goal is to maintain a sparse (bounded degree) expander topology despite heavy *churn* (i.e., nodes joining and leaving the network continuously over time). We assume that the churn is controlled by an adversary that has complete knowledge and control of what nodes join and leave and at what time and has unlimited computational power, but is oblivious to the random choices made by the algorithm.

Our main contribution is a randomized distributed protocol that guarantees with high probability the maintenance of a *constant* degree graph with *high expansion* even under *continuous high adversarial* churn. Our protocol can tolerate a churn rate of up to $O(n/\operatorname{polylog}(n))$ per round (where n is the stable network size). Our protocol is efficient, lightweight, and scalable, and it incurs only $O(\operatorname{polylog}(n))$ overhead for topology maintenance: only polylogarithmic (in n) bits needs to be processed and sent by each node per round and any node's computation cost per round is also polylogarithmic. The given protocol is a fundamental ingredient that is needed for the design of efficient fully-distributed algorithms for solving fundamental distributed computing problems such as agreement, leader election, search, and storage in highly dynamic P2P networks and enables fast and scalable algorithms for these problems that can tolerate a large amount of churn.

Keywords

dynamic network; P2P computing; distributed computation; randomized protocol; churn; fault-tolerance; expander graph

I. Introduction

Peer-to-peer (P2P) computing has emerged as one of the key networking technologies with many application systems, e.g., Skype, BitTorrent, CrashPlan, Symform, Bitcoin etc. For example, systems such as CrashPlan [1] and Symform [2] are relatively recent P2P-based storage services that allow data to be stored and retrieved among peers [3]. Such peer-based data sharing avoids costly centralized storage and retrieval and is inherently scalable. However, these systems are not fully P2P; they also use dedicated centralized servers in order to guarantee high availability of data — this is necessary due to the highly dynamic and unpredictable nature of P2P. Indeed, a key reason for the lack of fully-distributed P2P systems is the difficulty in designing highly robust and efficient algorithms for large-scale dynamic networks. Designing efficient and lightweight distributed computation protocols that are provably robust to the high amount of dynamism (where *both* nodes and edges change continuously over time) in a P2P network is thus an important step in realizing the goal of fully-autonomous decentralized computation even under the presence of large-scale dynamism and failures.

P2P networks are highly dynamic networks characterized by a high degree of *churn*, in which nodes continuously join and leave the network. Measurement studies of real-world P2P networks [4], [5], [6], [7] show that the churn rate is quite

John Augustine was supported by IIT Madras New Faculty Seed Grant, IIT Madras Exploratory Research Project, and Indo-German Max Planck Center for Computer Science (IMPECS).

Gopal Pandurangan was supported in part by NSF grant CCF-1527867.

Peter Robinson was partly supported by the European Community's Seventh Framework Programme (FP7/2007-2013) under the ASAP project, grant agreement no. 619706.

Scott Roche was supported in part by NSF grant CNS-0915985.

Eli Upfal was supported in part by NSF grant IIS-1247581.

high: nearly 50% of peers in real-world networks can be replaced within an hour. (However, despite a large churn rate, these studies also show that the total number of peers in the network is relatively *stable*.) P2P algorithms have been proposed for a wide variety of tasks such as data storage and retrieval [3], [8], [9], [10], [11], collaborative filtering [12], spam detection [13], data mining [14], worm detection and suppression [15], [16], privacy protection of archived data [17], and recently, for cloud computing services as well [2], [18]. However, all algorithms proposed for these problems have no theoretical guarantees of being able to work in a dynamically changing network with a very high churn rate. This is a major bottleneck in implementation and wide-spread use of P2P systems.

Motivated by the above considerations, several recent papers have taken steps towards designing provably efficient and robust algorithms for solving fundamental distributed computing problems in highly dynamic (with high churn) peer-to-peer networks. The work of [19] studies the fundamental distributed agreement problem in dynamic P2P networks with churn. Its main contribution is an efficient and scalable randomized distributed algorithm (i.e., each node processes and sends only polylogarithmic messages per round, and local computation per node is also lightweight) that guarantees stable almost-everywhere agreement[1] with high probability even under very high *adversarial* churn rate (up to linear in n *per round*, where n is the network size) in a *polylogarithmic* number of rounds. The work of [21] presented an efficient distributed algorithm for Byzantine agreement that works despite the presence of Byzantine nodes and high adversarial churn. This algorithm could tolerate up to $O(\sqrt{n}/\operatorname{polylog}(n))$ Byzantine nodes and up to $O(\sqrt{n}/\operatorname{polylog}(n)$ churn per round, took a polylogarithmic number of rounds, and was scalable. The work of [59] presented an efficient distributed algorithm for Byzantine leader election that could tolerate up to $O(n^{1/2-\varepsilon})$ Byzantine nodes (for a small constant $\varepsilon > 0$) and up to $O(\sqrt{n}/\operatorname{polylog}(n)$ churn per round, took a polylogarithmic number of rounds. The work of [22] focussed on the problem of storing, maintaining, and searching data in P2P networks. It presented a storage and maintenance algorithm that guaranteed with high probability, that data items can be efficiently stored (with only $O(1)$ copies of each data item) and maintained in a dynamic P2P network with churn rate up to $O(n/\operatorname{polylog}(n))$ *per round*.

A crucial ingredient that underlies all the above results is the assumption that though the topology — both nodes and edges — can change arbitrarily from round to round and is controlled by an adversary, the topology in *every round* is a (bounded-degree) *expander* graph. In other words, the adversary is allowed to control the churn as well as change the topology with the *crucial restriction* that it always remains an expander graph in every round. The assumption of an ever-present underlying expander facilitates the design of efficient algorithms that are highly robust (tolerating a large amount of churn per round). However, this is a very strong assumption, *that itself needs to be satisfied if one desires truly-distributed protocols that work under little or no assumption on the underlying topology*. This motivates designing a distributed protocol that actually *maintains* an expander topology under the presence of high adversarial continuous churn. Expanders have been used extensively to model dynamic P2P networks in which the expander property is preserved under insertions and deletions of nodes (e.g., see [23], [24], [25] and the references therein). However, none of these works guarantee the maintenance of an expander under the more challenging situation of high continuous adversarial churn. This is a fundamental ingredient that is needed to enable the applicability of previous results ([19], [21], [22], [59]) under a more realistic setting.

A. Our Main Result

Our main result is an efficient randomized distributed protocol that guarantees the maintenance of a *sparse* (bounded degree) topology with *high expansion* despite the presence of a large adversarial churn. The number of node changes *per round* is called the *churn rate* or *churn limit*. Though the churn rate can be large, we assume that the total number of nodes in the network is stable. We consider a churn rate of up to $O(n/\operatorname{polylog}(n))$, where n is the stable network size. We assume that the churn is controlled by an oblivious adversary that has complete knowledge of what nodes join and leave and at what time, but is oblivious to the random choices made by the algorithm. The adversary can remove any set of nodes up to the churn limit in *every* round. At the same time, it should add (an equal amount of[2]) nodes to the network with the following restrictions: (1) a new node should be connected to at least one existing node and (2) the number of new nodes added to an existing node should not exceed a fixed constant (thus, all nodes have constant bounded degree).

Our protocol (cf. Section III) is efficient, lightweight, and scalable, since the overhead for maintaining the topology is only polylogarithmic in n: requires only polylogarithmic in n bits to be processed and sent by each node per round and any node's computation cost per round is small (polylogarithmic in n). Our protocol guarantees that, despite a large adversarial churn, with high probability[3], there is a *large expander subgraph ($n - o(n)$ size)* present in the network in every round.

[1]In sparse, bounded-degree networks, an adversary can always isolate some number of non-faulty nodes, hence "almost-everywhere" is the best one can hope for in such networks [20].

[2]Our protocol can be adapted to work correctly as long as the number of nodes is reasonably stable over time, say, between $n - \kappa n$ and $n + \kappa n$ for some suitably small constant κ (for the same amount of churn per round, i.e., $O(n/\text{polylog } n)$).

[3]Throughout, with high probability (whp) means with probability at least $1 - n^{-c}$, for some constant $c \geqslant 1$.

The degree of each node is always bounded above by a fixed *constant*; this is useful for applications, as the number of connections (neighbors) of a node remains bounded and does not increase with the network size. The protocol assumes a short ($O(\log n)$ round) initial "bootstrap" phase, where there is no churn[4]; after this, the adversary is free to exercise its power to churn in/out nodes up to the churn limit. (A more detailed description is given in Section II.)

To the best of our knowledge, this is the first-known, fully-distributed protocol that guarantees expansion maintenance under highly dynamic and adversarial settings. Our algorithm is local (does not require any global topological knowledge), lightweight, and easy to implement. The protocol serves as a building block for enabling algorithms for fundamental distributed computing problems such as agreement, search and storage, and leader election in dynamic P2P networks. In particular, the expander maintenance protocol in conjunction with the protocols for agreement [19], and search and storage [22] enables a fully-distributed and autonomous protocol for these problems.

B. Technical Contributions and Overview

The main technical challenge that we have to overcome is maintaining good topological properties in highly dynamic networks where both nodes and edges can change by a large amount continuously. Indeed, when the churn rate is $O(n/\operatorname{polylog} n)$, in polylogarithmic number of rounds the entire network can be renewed! We would like distributed algorithms to work correctly and efficiently in such networks that keep *changing continuously over time* (not assuming any eventual stabilization or quiescence — unlike much of the earlier work on dynamic networks, e.g., [26], [27], [28], [29], [30], [31]). A technical contribution of this paper is showing how *random walks* can be used in a dynamic network to maintain an expander topology with high adversarial node churn (cf. Section IV). The basic idea is quite simple and is as follows: All nodes generate tokens (which contain the IDs of the respective source nodes) and forward each token via random walks continuously over time. These random walks, once they "mix" (i.e., reach close to the stationary distribution), reach essentially "random" destinations in the network. Thus the destination nodes receive a steady stream of tokens from essentially random nodes, thereby allowing them to sample nodes uniformly from the network. Crucially, the protocol uses the near-uniform distribution over the origin of the random walk tokens in the expander protocol: these provide random connections which lead to good expansion. While this basic idea is simple and clear, there are several technical challenges that have to be overcome to prove that the protocol actually works. The interaction between the actions of the adversary and the protocol (both determine what edges are added/deleted in every round) leads to complicated dependencies.

While it is easy to establish uniform sampling properties of a random walk in a static graph, this is no longer true in a dynamic network with adversarial churn — the churn can cause many random walks to be lost and also can introduce bias. Note that the protocol itself is continuously changing the network topology. This creates bias and dependencies in the sampling process; the sampling process in turn determines how connections are established in the protocol and thus determines the topology (which in turn determines how the tokens move subsequently). Another complication is that the adversary can isolate certain nodes (by churning out their neighbors) and thus the protocol has to make sure that any node that is cut off from the "core" network will quickly reconnect back; otherwise, the adversary can keep adding new nodes to this cut-off part leading to network fragmentation.Another challenge is that nodes can only try a *constant* number of re-connections per round, leading to a delay in connecting back to the core. This delay leads to some nodes getting isolated from the core; the adversary can then pile new nodes onto these isolated nodes, which do not get fresh tokens from the core. Moreover, new nodes that connect to the same old isolated node get the same copy of tokens; if these shared tokens are directly used for forming connections, then these edges will be correlated and hence not sufficiently random to achieve good expansion.

We describe some key features of our protocol (cf. Section III) that address the above challenges. First, as mentioned earlier, it crucially uses random walks to obtain a supposedly "near-uniform" distribution of IDs of nodes from a recent past. However, it is not clear, whether the "near-uniform" property is true for all tokens, even for those that appear mixed, i.e., have walked for $\Theta(\log n)$ steps. The protocol uses a simple *local* criterion to decide how well connected it is to rest of the network: if its number of received tokens is below a certain threshold, then it decides that all of its tokens and present connections are not good and tries to "refresh" its edges. On the other hand, nodes that do get sufficiently many tokens also refresh their connections, but with small probability; this is essential to maintain a graph that is random enough.

To simplify the analysis of the protocol, we follow a two-step proof idea (cf. Section IV). First, we define a stochastic graph process called Υ-Process (cf. Section IV), that has to satisfy some properties; these properties capture the actions of the adversary and the protocol, but it *abstracts away* the details of the distributed expander protocol. We show that the

[4]Without a bootstrap phase, it is easy to show that the adversary can partition the network into large pieces, with no chance of forming even a connected graph.

Υ-process produces a sequence of graphs, each of which contains a large expander subgraph[5] with high probability. In the second step, we show that the expander protocol produces a sequence of graphs that conforms to this Υ-process. Our proof crucially uses a technical result (cf. *Sampling Lemma* — Lemma 6) that shows that most random walks have the usual desirable properties (near-stationary distribution and almost uniform origins) even in a highly dynamic adversarial network. We note that our technique can handle churn up to $O(n/\operatorname{polylog} n)$. Informally, this is due to the fact that at least $\Omega(\log n)$ rounds are needed for the random walks to mix, before which any non-trivial computation can be performed. This seems to be a fundamental limitation of a random walk based method.

C. Other Related Work

There has been a lot of work on P2P protocols for maintaining desirable properties (such as connectivity, low diameter, high expansion, bounded degree) under churn (see e.g., [25], [32], [34] and the references therein), but these do not work under large continuous adversarial churn. On the other hand, an important aspect of our protocol is that it works even when the network is continually changing in an highly dynamic and adversarial manner. Most prior algorithms (e.g., [24], [35], [36], [23], [37], [38]) will only work under the assumption that the network will eventually stabilize and stop changing or there is a "repair" time for maintenance when there are no further changes (till the repair/maintenance is finished); these algorithms do not work under high continuous adversarial churn. Law and Siu [24] provide a distributed algorithm for maintaining an expander in the presence of limited number of insertions/deletions; their algorithm does not work for high continuous adversarial churn. In [36] it is shown how to maintain the expansion property of a network in the self-healing model (there are no changes during repair) where the adversary can delete/insert *one* node in every step. In the same model, [23] present a protocol that maintains constant node degrees and constant expansion (both with probability 1) against an adaptive adversary, while requiring only logarithmic (in the network size) messages, time, and topology changes per deletion/insertion. [35] presented a self-stabilizing algorithm that converges from any weakly connected graph to a SKIP graph (which is an expander with high probability) in time polylogarithmic in the network size. In [37] the authors introduce the hyperring, which is a search data structure supporting insertions and deletions, while being able to handle concurrent requests with low congestion and dilation, while guaranteeing $O(1/\log n)$ expansion and $O(\log n)$ node degree. The k-Flipper algorithm of [38] transforms any undirected graph into an expander (with high probability) by iteratively performing flips on the end-vertices of paths of length $k+2$. Based on this protocol, the authors describe how to design a protocol that supports deletions and insertions of nodes. There has also been significant prior work in designing P2P networks that are provably robust to a large number of Byzantine faults (e.g., see [39], [40], [41], [42], [43]). These focus on robustly enabling storage and retrieval of data items under adversarial nodes. However, these algorithms will not work in a highly dynamic setting with large, continuous, adversarial churn. The work of [33] presents a solution for maintaining a clustering of the network where each cluster contains more than two thirds honest nodes with high probability in a setting where the size of the network can vary polynomially over time; however the churn is limited to $O(\operatorname{polylog} n)$ per round and the network degree is also polylogarithmic in n. The work of King et al. ([44]), which achieves distributed Byzantine agreement and leader election in *static* P2P networks, raised the open question of whether one can design robust P2P protocols that can work in highly dynamic networks with a large adversarial churn.

Kuhn et al. consider in [34] that up to $O(\log n)$ nodes (adversarially chosen) can crash or join per constant number of time steps. Under this amount of churn, it is shown in [34] how to maintain a low peer degree and bounded network diameter in P2P systems by using the hypercube and pancake topologies. There has been lot of work on dynamic network models where *only the edges* change over time but the nodes remain *fixed*, see e.g., [45], [46], [47]; these works do not apply to the churn setting considered here. Scheideler and Schmid show in [48] how to maintain a distributed heap that allows join and leave operations and, in addition, is resistent to Sybil attacks. A robust distributed implementation of a distributed hash table (DHT) in a P2P network is given by [43], which can withstand two important kind of attacks: adaptive join-leave attacks and adaptive insert/lookup attacks by up to εn adversarial peers. This paper assumes that the good nodes always stay in the system and the adversarial nodes are churned out and in, but the *algorithm* determines where to insert the new nodes. Naor and Weider [41] describe a simple DHT scheme that is robust under the following simple random deletion model — each node can fail independently with probability p. They show that their scheme can guarantee logarithmic degree, search time, and message complexity if p is sufficiently small. Hildrum and Kubiatowicz [40] describe how to modify two popular DHTs, Pastry [49] and Tapestry [50] to tolerate random deletions. Several DHT schemes (e.g., [51], [52], [53]) have been shown to be robust under the simple random deletion model mentioned above. There also have been works on designing fault-tolerant storage systems in a dynamic setting using quorums (e.g., see [54], [55]). However, these do not apply to our model of continuous churn.

[5]This is essentially the best that can be shown, since at every point in time there can be a small set of nodes that are effectively isolated from the main part of the network.

II. PRELIMINARIES

The Adversarial Network Model: We consider a synchronous dynamic network controlled by an oblivious adversary. The adversary fixes a sequence of graphs $\mathcal{H} = (H_1, H_2, \ldots, H_i, \ldots)$ with each H_i defined on a vertex set V_i. The size of the vertex set is assumed to be stable, i.e., $|V_i| = n$ for all i, and $|V_i \setminus V_{i+1}| = |V_{i+1} \setminus V_i| \in O(n/\log^k n)$, where k is a constant that will be fixed in the analysis[6]. The vertex set V_i refers to the set of nodes present in the network in round i. Each node has a unique ID chosen from an ID space of size polynomial in n. The edges must be viewed as a rudimentary set of connections provided by the adversary. The degree of each graph H_i is bounded by a constant denoted by $\deg(\mathcal{H})$.

For the first $B = \beta \log n$ rounds (for a sufficiently large constant $\beta > 0$), called the *bootstrap phase*, each H_i ($1 \leqslant i \leqslant B$) is a (an arbitrary) constant degree expander with *expansion at least* $\alpha > 0$ [7], a fixed constant. In the bootstrap phase (i.e., during the first B rounds), the adversary is *silent*, i.e., there is no churn and no edges are changed, and hence all the H_i's are identical. We can think of the bootstrap phase as an initial period of stability during which the protocol prepares itself for a harsher form of dynamism.[8]

After the bootstrap phase, the requirements on the adversary are significantly reduced. It is only required to ensure that any new node u that enters at, say, round i, must be connected in H_i to at least one other pre-existing node v, i.e., if $u \in V_i \setminus V_{i-1}$, then, u must be connected to some $v \in V_{i-1} \cap V_i$ in round i. The adversary must also ensure that the degree bound $\deg(\mathcal{H})$ is not violated. The following subtleties are noteworthy. Firstly, we only require that a new node is connected to a pre-existing node in its very first round upon entering the network. Secondly, after the bootstrap phase we don't even require each H_i to be connected — in fact, the only required edges in H_i are the edges connecting new nodes to pre-existing nodes.

Our goal is to design a distributed protocol to maintain a graph process

$$\mathcal{G} = (G_1, G_2, \ldots, G_i, \ldots)$$

over time such that each G_i, for $i \geqslant B = \Theta(\log n)$ (i.e., *after* the bootstrap phase), has good expansion, while maintaining a constant degree bound Δ with high probability. The protocol must achieve this goal by dynamically adding/deleting edges over time. Each $G_i = (V_i, E_i)$ has the same vertex set as that of H_i, while the edges E_i (separate from the edges of H_i) are determined by the actions of the adversary and the protocol. In each round i, the protocol can add a set $E_i^{\downarrow}$ of new edges. Also, a set $E_i^{\uparrow}$ of edges that were added by the protocol in some previous round is lost — some explicitly removed by the protocol and others removed implicitly because vertices to which they were incident were churned out. Thus, each $E_i = (E_{i-1} \setminus E_i^{\uparrow}) \cup E_i^{\downarrow}$. ($E_1$ is empty). We emphasize that the graph H_i is only presented at round i and the protocol is unaware of future graphs.

Communication Model: Communication is via message passing. If a node u knows the ID of another node v either directly because u and v are neighbours in the current graph in $\mathcal{G}$, or because they are neighbors in H_i (assuming we are in round i), or indirectly through received messages, then u can communicate with v[9]. Each node can send and receive messages of size polylogarithmic (in n) bits. Furthermore, each node is restricted to some *constant* $M = \Delta + \deg(\mathcal{H})$ number of messages it can send and receive (and therefore, the number of nodes to which it can send and receive) in a single round.

Notice that a sender u can ensure that it sends out at most M messages intended for at most M recipients. However, due to the distributed and parallel nature of the senders, they may not be able to ensure that the number of messages intended for a single node is within the bound M. If more than M nodes send messages intended for v, it can only receive M messages, while others must be dropped. We assume the following priority by which messages are either accepted or dropped:

1) All messages (say, of cardinality $j \leqslant M$) sent to v along edges in $\mathcal{H}$ and $\mathcal{G}$ are received successfully, and,
2) among the remaining messages intended for v, only an arbitrarily chosen subset of $M - j$ messages reach v; the rest are all dropped.

Adding/Deleting Edges in $\mathcal{G}$: We assume the existence of a simple two-step "handshake" protocol (described in the next paragraph) that allows any node u to propose an edge between itself and another node v (provided u knows v's id); the node v, in turn, can either explicitly accept or ignore (thereby implicitly rejecting) the proposal. Furthermore, a node u can drop any edge (u, v) incident to it without consulting v; we assume that v will be immediately notified that (u, v) has been dropped.

[6]For the sake of clarity, we do not seek to optimize constants and $\log n$ factors. Our analysis shows that k need not be > 3.

[7]This means that, for any subset S of up to $n/2$ nodes in the graph H_i, the number of edges across the cut given by S and the remaining nodes, is at least $\alpha|S|$.

[8]We note that in the unlikely event that our protocol fails in some round (that can happen with at most $1/n^c$, for sufficiently large c), then a bootstrap phase is needed to get it started again, before it can handle adversarial churn again.

[9]This is a typical assumption in the context of P2P and overlay networks, where a node can establish communication with another node if it knows the other node's IP address.

Sequence of Events in a Round: We now describe the precise sequence of events that comprise a single round i in our model:

1. The adversary presents the new graph H_i in which some of the old nodes have been churned out and some new nodes have been churned in (assuming round i occurs after the bootstrap phase). The graph G_{i-1} loses all edges that were incident to nodes that were churned out.

2. Each node is then allowed to perform some local computation including private coin flips and send messages of size polylogarithmic in n to up to $M = \Delta + \deg(\mathcal{H})$ other nodes. This communication step, among other purposes, is particularly useful for nodes to send requests for adding edges.

3. Nodes can again perform some more local computation, and again send messages of size polylogarithmic in n to up to M (a constant, defined above) other nodes. The messages sent in this step are typically responses to messages sent out in the previous step. For instance, a node v that had received a request from a node u to form an edge could either send an accept message or ignore it (implicitly rejecting the request).

4. Once v accepts node u's edge request, edge (u, v) is added to the communication graph. The communication graph now includes all the newly added edges, and is designated G_i.

We note that a round is comprised of a two-step protocol to add an edge: the requesting node sends a message to the other node, which takes one more step to accept/reject. The above assumption is slightly different from the classic notion of a round, where messages are sent only one way per round (hence implementing these two steps will actually take two rounds). This assumption is not just a mere technicality; on the other hand it is needed. Using an information flow argument, we prove (cf. Theorem 2) that there is no protocol in the classic synchronous round model (cf. [56]) that can maintain connectivity (let alone expansion) for a large majority of the nodes, even if the churn is limited to $\Theta(\sqrt{n \operatorname{polylog}(n)})$ nodes per round (cf. Section V).

III. Expander Maintenance Protocol

A. High-level Overview and Intuition

Our protocol works by ensuring two interdependent invariants (whp). The first invariant is that the nodes can sample from the set of IDs in the network almost uniformly at random. We implement this sampling via *random walks*. For the sampling technique to work efficiently, we need our second invariant, namely, that the network maintains good expansion, and for this, we employ a technique of connecting each node to a constant number of other nodes chosen (almost) uniformly at random, thus bringing us back to our first invariant. With the two invariants being so intimately dependent on each other, the bootstrap phase in which expansion is guaranteed is crucial in helping the protocol to get started. During this phase, the random walks based sampling procedure gets started, which, in turn, allows the network to continue maintaining good expansion beyond the bootstrap phase. While this high level idea is quite straightforward, there are some significant challenges (cf. Section I-B) to be overcome in order to get the protocol to work.

Before we go into the protocol and its analysis in detail, we give a high-level overview of the main ideas and some intuition into understanding its design (cf. Section III-B). Since nodes have to maintain bounded degree, and since each node requests as well as accepts connection requests from other nodes, it is helpful to partition the degree set (with respect to a particular node) into two sets — *red* and *blue* — where the number of blue edges will always be *at most* a constant fraction $(1/6)$ of the maximum number of edges (Δ). Blue edges of a node are those that it requests and red edges are the ones that it accepts; note that a red edge of some node will be a blue edge with respect to some other node. The *main invariant* maintained by the protocol is *the number of blue edges per node*; if this number falls below a certain threshold δ (due to loss of blue edges caused by churn), then the protocol will try to regain the lost blue edges. The mechanism for creating new blue edges is as follows. Each node maintains a buffer (called the prioritized token buffer) that contains only "mature" tokens (i.e., well-mixed tokens) prioritized by time (recent is higher priority). When a node wants to create a blue edge it will choose one of these tokens and contact the node whose ID is contained in the token. An important idea in the protocol is that a node may reconnect even though it may *not have lost any blue edge*. The protocol uses a simple mechanism to decide when to reconnect: when the number of mature tokens it receives falls below a certain threshold, then it decides that it is not well-connected to the rest of the network and refreshes its connections. The key Sampling Lemma (Lem. 6) shows that this local criterion works; it also shows that most tokens are well-mixed (i.e., they are distributed in a near-uniform fashion and hence the endpoints of the blue edges are as well) despite the actions of the adversary and the protocol.

As mentioned earlier, a main intuition behind the protocol is that it tries to maintain a graph that resembles a random (bounded-degree) graph in every round whp. Thus, in addition to reconnections that are caused due to lost blue edges, in every round, every node with probability $\Theta(1/\text{polylog } n)$, simply drops all its blue edges and reconnects again (cf. "Normal mode" in Algorithm III.2 (line 7)). This refreshing of connections creates new "random enough" edges continuously in the network, which is (also) useful in showing expansion properties in every round. Since the number of blue edges is

significantly less than the number of red edges there is a good (a constant) probability of establishing a connection in a round (cf. Lemma 1).

A complication that the protocol needs to handle is that since nodes can accommodate only constant number of connections, many connection requests are turned down in a round (which occurs with constant probability); thus many nodes that lose connections may remain "cut-off" (i.e., isolated from the "core" network that has size $n - o(n)$ whp) for a while. The adversary can attach new nodes to these cut-off nodes; these new nodes will not get enough mature tokens, since the cut-off nodes themselves are starved of tokens. This necessitates *sharing* of tokens between the new nodes and the cut-off nodes (cf. "Reconnect mode" in Algorithm III.2); such sharing of tokens is problematic if continued for a long time (since tokens are no longer independent among nodes). To address this, the protocol allows these cut-off nodes to contact nodes using these shared tokens (called *stale tokens*) to obtain fresh mature tokens that are mixed well. We show in our analysis (cf. proof of Lemma 7), that nodes cannot remain cut-off for long and will reconnect to the "core" network in at most $O(\log n)$ rounds (whp).

This pseudocode is executed in every round i by each node v. Recall that each round consists of two communication steps, but this procedure only requires one communication step. WLOG, we assume the first communication step is used, while the second is ignored. A *mature random walk token* is one that has walked for at least $\tau \in \Theta(\log n)$ rounds.

Initiate new random walk tokens.

1: Node v initiates $z \in \Theta(\log^{2k+1} n)$ random walk tokens containing v's id.

Perform random walk step for tokens not yet matured.

2: During the bootstrap phase, send each immature token to a random neighbour in H_i. Skip to line number 13.
3: **if** v has self loops in G_i **then**
4: With probability $\theta/\log^{k-1} n$ mark an arbitrary self loop port p incident to v in G_i. (With the remaining probability $1 - \theta/\log^{k-1} n$, no incident self loop is marked.)
5: // The constant θ must be large enough to ensure (whp) that the total number of marked self loop ports in the network exceeds the total number of dangling blue ports.
6: **for all** tokens t at v **do**
7: Choose a port p (in G_i) uniformly at random from among the Δ ports.
8: **if** p is either a dangling port or a marked port **then**
9: Eliminate t.
10: **else**
11: Send t along p (in communication step 1).
12: Unmark any marked port incident to v.

Handle mature tokens.

13: **if** the number of mature tokens v received is at least $(1 - \eta)z$ for a small fixed constant $\eta > 0$ **then**
14: // The above condition is useful because it is indicative of how well v is connected in the network.
15: // Note that this condition is also used in the expander protocol.
16: Node v inserts all mature tokens into its prioritized buffer. Designate these tokens as fresh.
17: **else**
18: All mature tokens received in the current round are discarded.

Algorithm III.1: Distributed Protocol for Random Walk-Based Sampling

B. Detailed Description of the Protocol

Degree Bound, Red Edges, Blue Edges: As stated in Section II, each node in G_i, for all i, must have an upper bound of Δ on its degree. Each node partitions its incident edges into 2 types: red and blue edges. In fact, every edge in G_i will be blue at one of its ends (in particular, the end that proposed its creation) and red at the other. (Note that in G_1, the red and blue edges are empty — not yet created). Each node u seeks to have δ (where $\delta < \Delta/6$) *blue edges*, i.e, edges that are created upon u's proposal. As will be elaborated further, if u has fewer than δ blue edges at any time, then it will propose more edge connections in an attempt to meet this quota. Out of the Δ ports at each vertex, the protocol will set aside δ

This pseudocode is executed every round by each node v. Recall that each round consists of two communication steps. We tag them as CS1 and CS2, respectively. Node v can either be in `normal` mode or `reconnect` mode. A node that has newly entered the network (or in the bootstrap phase) is in `reconnect` mode until it manages to find δ blue edges. Once it gets δ blue edges it switches to `normal` mode. The Random Walk Sampling procedure (cf. Algorithm III.1) is assumed to be running simultaneously.

Whenever Node v is in `normal` mode

1: // The random walks based sampling procedure operates in CS1.
2: CS1: Receive requests for edges and tokens sent by nodes in `reconnect` mode.
3: **if** the number of mature tokens received in current round is $< (1-\eta)z$ (cf. Line 13 of Algorithm III.1) **then**
4: Drop all incident blue edges and enter `reconnect` mode.
5: CS2: Accept as many edge requests as possible without exceeding $\Delta - \delta$ red edges.
6: CS2: Provide $\Theta(\log n)$ tokens (if available) to each node that requests tokens. Designate them as fresh tokens.
// Always ensure that $\Theta(\log n)$ tokens are available for v's own consumption.
7: With probability $1/\log^k n$:
(1) drop all incident blue edges,
(2) and enter `reconnect` mode.
8: **if** the number of blue edges is $< \delta$ **then**
9: Enter `reconnect` mode.

Whenever Node v is in `reconnect` mode

10: Ignore (and thus implicitly reject) any edge requests.
11: **if** Node v is new in the network **then**
12: CS1: Request tokens from neighbours in H_i.
13: CS2: Collect tokens sent by neighbours and store in the prioritized token buffer.
14: **else if** Node v has been in the network for more than one round **then**
15: **if** Fresh tokens are available in reserve **then**
16: CS1: Use the highest priority fresh tokens in the prioritized token buffer to make up to δ edge requests.
17: **else if** Stale tokens are available in the prioritized token buffer **then**
18: CS1: Use $O(1)$ stale tokens chosen uniformly at random to request for fresh tokens.
19: **else**
20: Do nothing. // This situation arises in the bootstrap phase before tokens have mixed.
21: CS2: Provide tokens to each node that requests tokens: either c fresh tokens if available or all the stale tokens (but never giving out the top c tokens which are always for v's own use).
22: **if** the number of blue edges is δ **then**
23: Enter `normal` mode.

Algorithm III.2: Distributed maintenance protocol

ports — called blue ports — for blue edges. Any blue port that does not have an incident blue edge is called a *dangling port*. A red port that does not have an incident red edge to another node will have a *self-loop* to itself (i.e., the edge will leave and end at the same port). The remaining $\Delta - \delta$ available ports are reserved for red edges and self loops. Ports and self-loops prove useful later in the analysis of the protocol as they can be used to ensure regularity (cf. Section IV). Notice that there can be at most δn blue edges, and therefore the same number of red edges, at any point in time. This ensures that at least $0.8n$ nodes are guaranteed to have spare red ports at any time (this will be used later in the analysis (cf. Lemma 1) to bound the probability of forming blue edges.)

We first describe a random walks based sampling procedure before introducing the details of our main expander maintenance protocol. Although this sampling procedure is executed in tandem with the expander maintenance protocol, we describe this in isolation for the sake of clarity. Algorithm III.1 provides the details in pseudocode format.

Random Walks Based Sampling: Each node u generates $z \in \Theta(\text{polylog } n)$ (more precisely $z = \Theta(\log^{2k+1} n)$, where k is the constant in the churn limit) *tokens* in each round t. Each token contains the following information: the ID of u that

generated it, as well as a counter to keep track of how many rounds have passed since the token has been generated. For all tokens, the counter is initially set to zero, and in every round the node that currently holds the token is responsible for incrementing its counter by one and forwarding it to one of its neighbors chosen uniformly at random. In other words, each token is taking a random walk on the (possibly changing) edges of the network.

Each random walk token walks for up to $\tau = \Theta(\log n)$ rounds (τ is called the *mixing* time — cf. Lemma 5), after which it is called a *mature* token. The number of mature tokens received by a node is indicative of whether they can be used for sampling nodes (cf. Lemma 6). Thus, if a node receives fewer than $(1-\eta)z$ (for a sufficiently small fixed constant $\eta > 0$) mature tokens in any round, those mature tokens are discarded. Otherwise, they are held in a prioritized token buffer for consumption as samples for the next (at most) $\Theta(\log n)$ rounds (thus a mature token stays in the current node for at most $O(\log n)$ rounds till it is used by the current node to reconnect or transferred to another node due to some request). Whenever a random node sample is required, the highest priority token from the buffer is picked (without replacement) and the token's starting location is used as a random sample.

Prioritized Token Buffer: Each node v has a prioritized token buffer of capacity $C \in \Theta(\text{polylog } n)$ for mature random walk tokens. Each token that attains maturity in v is designated *fresh* and placed in the prioritized token buffer (possibly after evicting a token of least priority). We assume for simplicity that $(1-\eta)z \geqslant C$, so a node can fill its buffer in a single round when it receives $\geqslant (1-\eta)z$ that can be used as samples. Sometimes, under a dearth for fresh tokens (this can happen when nodes are "cut-off" as mentioned above in Section III-A), some tokens are copied and the copies are shared among several nodes; such copied tokens are designated as *stale*. Fresh tokens are prioritized in the buffer over stale ones. Between two tokens that are either both fresh or both stale, the token that matured most recently is prioritized. The top $c = \Theta(\text{polylog } n)$ tokens in the buffer are set aside for v's own consumption and are never shared; here, c is chosen such that C/c is a sufficiently large. When there are at least $C/2$ tokens in the buffer, up to c fresh tokens are given out to requesting nodes. When there are fewer than $C/2$ tokens, upon request for tokens, all tokens (except the top c tokens) are designated stale and copies of the stale tokens are sent to the requester; note that in this case, the sender retains copies of the stale tokens.

Distributed expander maintenance protocol: Our distributed maintenance protocol will run on all nodes in the network, and is initiated either at the beginning by each node present in V_1, or by a churned-in node when it enters the network. Note that the random walk sampling process as described above is running in the background to ensure the continued forwarding of tokens and enabling each node to maintain a pool of matured tokens. Once the initial bootstrap phase has passed, the adversary is allowed to make its proscribed changes to the network and the maintenance protocol will seek to maintain a dynamic network $\mathcal{G}$ in which each G_i has good expansion.

We now describe the expander maintenance protocol from the perspective of a single node v, in particular, focussing on the steps it must execute in each round. The node v can be in one of two modes depending on the number of the blue edges it currently has: `normal` (has δ blue edges) or `reconnect` ($< \delta$ blue edges). The precise protocol in pseudo code format is provided in Algorithm III.2.

The `Normal` *Mode:* The protocol starts with a check to see how many mature tokens v has received, which, as we shall see, is indicative of how well v is connected to the rest of the network. If the number of such tokens is below the required threshold of $(1-\eta)z$, it will drop/delete all of its existing blue edges and switch into `reconnect` mode during which it will take the steps required to regain lost connections.

If, on the other hand, the number of mature tokens received is sufficient, i.e., at least $(1-\eta)z$, then with probability $1-1/\text{polylog } n$ the node will continue in normal mode. With probability $1/\text{polylog } n$, however, all the blue edges are dropped and the node v will then switch to `reconnect` mode. This will ensure (whp) that no edge is more than $O(\text{polylog } n)$ rounds old in the network.

The `Reconnect` *Mode:* Node v enters `reconnect` mode whenever fewer than the stipulated number of blue edges are incident to v, and needs to regain new connections. This situation arises under two circumstances: when the node has just arrived and has not had a chance to establish connections, and when the node has lost some incident blue edges.

Note that a node $v \notin V_1$ arrives only after the bootstrap phase as there is no churn during the bootstrap phase. As described in Section II, the adversary chooses an existing node w to which the new node v is initially connected to. Upon connecting, the existing node w will send v a package of $O(\text{polylog } n)$ number of tokens. These tokens can be marked as "fresh" or "stale". Receiving "stale" tokens implies that node w itself was running low on fresh tokens, and thus needed to copy tokens it already had to satisfy v's request. As such, stale tokens affect the probability distribution of the nodes they contain and create the possibility that too many nodes will have copies of the same tokens. We prove that the latter does not occur, but the former requires another step to be added to the protocol: If v receives copied stale tokens, it first contacts one or more nodes (drawn uniformly at random from those stale tokens) and requests a package of mature fresh tokens from them. If the recipient node has fresh tokens to spare (i.e., without copying), it will send them. This intermediate step ensures that v uses fully mixed mature tokens to seek new edges. Thus, upon receipt of those fresh tokens (or if the original tokens handed to v

were already fresh), v makes edge requests until it receives δ blue edges.

IV. The Υ-Process and the Analysis of the Protocol

The graph process $\mathcal{G}$ output by the expander maintenance protocol is the result of the interaction between two different entities, namely, the adversary and the expander maintenance protocol itself. Furthermore, the protocol has two aspects to it that further interact with each other. The first is the random walks based sampling (cf. Algorithm III.1) and the second being the aspect of actually adding and deleting edges to ensure expansion (cf. Algorithm III.2). In order to aid in the analysis of our algorithm, we unify these interacting computational entities into one mathematically coherent family of processes. A member in this family is called an Υ-process and its behavior is the combination of (i) the actions of a particular adversary, (ii) a particular choice of acceptable values for various parameters, and (iii) random bits available to the Υ-process. Being a mathematical construction, we now present the Υ-process as a broad sequence of mathematical steps without delving into the algorithmic mechanics by which those steps are implemented, thereby allowing us to focus on the mathematical essentials needed for the analysis.

Let $\mathtt{G}_1$ be an arbitrary graph with node degree $\leqslant \Delta$ that has a subgraph of size $n - O(n/\log^k n)$ that has expansion at least α.
for $i = 2, 3, \ldots$ **do**
  Arbitrarily choose a churn out set $C_{out} \subset \mathtt{V}_{i-1}$ of cardinality at most $O(n/\log^k n)$.
  Create a new set C_{in} of vertices of equal cardinality as C_{out}.
  Establish vertex set $\mathtt{V}_i = (\mathtt{V}_{i-1} \setminus C_{out}) \cup C_{in}$.
  $\mathtt{E}_i \leftarrow \{e = (u, v) \in \mathtt{E}_{i-1} | u \in \mathtt{V}_i$ **and** $v \in \mathtt{V}_i\}$ // Inherit edges from $\mathtt{G}_{i-1}$ induced by $\mathtt{V}_i$.
  Create a graph $\mathtt{G}_i = (\mathtt{V}_i, \mathtt{E}_i)$. // We will need to modify $\mathtt{G}_i$ a bit more before finalizing it.

  For every $v \in \mathtt{V}_i$, initialize $B(v) = \delta$. // $B(v)$ is the number of blue edges v must have in $\mathtt{G}_i$.
  Choose $\mathtt{V}^* \subseteq \mathtt{V}_i$ to be the set of vertices with fewer than δ incident blue edges such that:
    1) $C_{in} \subseteq \mathtt{V}^*$,
    2) $|\mathtt{V}^*| \in O(n/\log^{k-1} n)$, and
    3) no node has been chosen as a member of $\mathtt{V}^*$ for a contiguous period of $\Theta(\log n)$ rounds.
  for all $v \in \mathtt{V}_i \setminus \mathtt{V}^*$ **do**
    With probability in $\Theta(1/\log^k n)$ delete all blue edges incident at v and add v to $\mathtt{V}^*$.
    // This ensures (whp) that no blue edge is more than $O(\text{polylog } n)$ rounds old.
  for all $v \in \mathtt{V}^*$ **do**
    blue = the number of blue edges currently incident to v.
    Set $B(v)$ to an arbitrary value in $[\text{blue}, \delta]$.

  while $\exists v \in \mathtt{V}_i$ with fewer incident blue edges than $B(v)$ **do**
    Arbitrarily pick a vertex v with fewer incident blue edges than $B(v)$.
    Arbitrarily select a set $V' \subseteq \mathtt{V}_i \setminus \{v\}$ of cardinality at least (say) $0.8n$, where $n = |\mathtt{V}_i|$, such that every vertex in V' has at the moment strictly fewer than $\Delta - \delta$ incident non-self-loop red edges.
    Define an almost uniform probability distribution D_u over all $u \in \mathtt{V}_i \setminus \{u\}$ in the following sense:
      1. $D_u \leqslant 1/n + 1/n^c$, for a suitably large constant c.
      2. If the number of red edges is $\Delta - \delta$, then set $D_u = 0$.
      3. If $u \in V'$, then D_u must be in $\Omega(1/n)$.
      // Of course, $\sum_{u \in \mathtt{V}_i \setminus \{v\}} D_u$ must equal 1.
    A node u is drawn from D_u and a new edge (v, u) (that is blue at v and red at u) is added.

  Place a self loop on every unused port reserved for red edges.
    // Note that unused ports reserved for blue edges are left dangling.
  The graph $\mathtt{G}_i$ is now finalized.

Procedure IV.1: The Υ-Process

A. Overview and Intuition

An Υ-process generates a sequence of graphs $\mathbf{G} = (\mathtt{G}_1, \mathtt{G}_2, \ldots, \mathtt{G}_i, \ldots)$. (For notational clarity, we use $\mathtt{G}_i = (\mathtt{V}_i, \mathtt{E}_i)$ to denote a graph generated by the Υ-process and G_i to denote graphs generated by the expander maintenance protocol.)

The Υ-process is presented in detail in Procedure IV.1 on page 10. Before we go into the analysis, we present an overview of the process and intuition behind its analysis. The main intuition behind the process is that it captures the actions of the distributed maintenance protocol and the adversary.

The Υ-process starts off with being a bounded degree graph that contains a large expander subgraph—this captures the graph formed at the end of the bootstrap phase of the protocol which lasts for $\Theta(\log n)$ steps. The goal is to show that (i) the sequence of graphs $\mathbf{G}$ generated by the Υ-process has the required expansion properties (cf. Lemma 2) and that (ii) the execution of the expander maintenance protocol (starting from the end of the bootstrap phase) is an Υ-process (cf. Lemma 7). In this regard, for all $i \geqslant 1$, the properties of $\mathtt{G}_i$ (the graph output by the Υ-process in round i) will correspond to G_{i+B}, the graph output by the expander maintenance protocol in round $i + B$. In particular, each $\mathtt{G}_i$ will have degree at most Δ, its edges will be partitioned into red and blue edges (in the same proportion) as in G_{i+B}. The process captures the adversarial churn and the actions of the protocol. The lines 2-7 follow the actions of the adversary where, in each round, a set of $O(n/\text{polylog } n)$ nodes (determined by the adversary) are churned out and an equal number is churned in. The edges that are not destroyed due to churn are carried over from the round to the next. On the other hand, if a blue edge is lost, it is established again in a way that mimics the protocol. Recall that in the expander maintenance protocol, when a node has fewer than δ blue edges, it tries to connect until it succeeds (or is churned out). Thus, it remains deficient of blue edges until some round and then gets an edge chosen uniformly at random. This is captured in the Υ-process in the following generic way. The Υ-process delineates the two decisions, namely, (i) when to add an edge and (ii) how to add an edge. From lines 8 until line number 15, the Υ-process is required to ensure the conditions (listed under line number 9). These lines do not actually create the edges, but rather place budgets on the number of blue edges that need to be added to each node. This budget can be set to an appropriate value (with δ being the maximum possible number of blue edges) that captures the number of blue edges that a node has in the particular round of the protocol. The conditions specify the set of nodes V^* (of size $O(n/\text{polylog } n)$) that have less than δ blue edges in the round comprising: (1) newly churned in nodes; (2) nodes that choose to refresh their blue edges (with probability $O(1/\text{polylog } n)$ as in the protocol). No node is in V^* for a contiguous period of $\Omega(\log n)$ rounds — this condition crucially captures the property that every node will regain its lost full quota of blue edges in $O(\log n)$ rounds; we show that the protocol satisfies this condition whp (cf. Claim 1).

After the budgets are set, the edges are actually created at a later part (from line number 16 to 20) of the Υ-process definition where the endpoints are chosen almost uniformly at random from a large set of vertices. Since the RW-Sampling algorithm has been successfully running up until round $i - 1$ (as a consequence of our assumption that the expander maintenance protocol has been an Υ-process until time step $i - 1$ in conjunction with Lemma 6), the expander maintenance protocol also chooses the red end from the current set of vertices based on a distribution that is almost uniform over a large fraction of current (i.e., round i) vertices.

Our analysis proceeds as follows. First we show that the graphs generated by the Υ-process contain a large-sized $(n - o(n))$ expander graph whp. Showing this requires the following main ingredients:

(1) Show that the random walk sampling will generate near-uniform random samples from a large-sized subset of the nodes (Section IV-C). The Sampling Lemma (Lem. 6) shows this. For technical reasons it is difficult to work with the Υ process graph $\mathtt{G}_i$, since this graph is not regular, and more importantly, this graph is dynamically changing in every round (and mixing takes $\Theta(\log n)$ rounds). For the *purpose of analysis*, we convert the graph $\mathtt{G}_i$ to $\bar{\mathtt{G}}_i$ (Algorithm IV.2). The new process $\bar{\mathtt{G}}_1, \ldots \bar{\mathtt{G}}_i, \ldots$ is regular (i.e., all nodes have Δ degree with no missing edges) and there is no churn in this process. Regularity is obtained by introducing the so called *ghost edges* (which are simply virtual edges that take the place of missing edges in $\mathtt{G}_i$ — they are determined in lines 10-14 of Algorithm IV.2) which allow us to apply techniques from [57]. Tokens that travel through ghost edges are lost (and ghost edges are adversarially determined, since missing edges are) and some bias is also introduced. (Nevertheless, the Sampling Lemma shows that these are within bounds and we obtain the desired sampling result.)

(2) Since $\mathtt{G}_i$ is obtained by (adversarial) deletion of $\Theta(n/\text{polylog } n)$ nodes and edges from $\bar{\mathtt{G}}_i$, we show that this still leaves a large expander subgraph in $\mathtt{G}_i$, the Υ graph.

(3) The key Sampling Lemma shows the near-uniform distribution of tokens in the Υ process graph $\mathtt{G}_i$, by appealing to the distribution of real tokens (not ghost tokens) in $\bar{\mathtt{G}}_i$.

We then show (in Section IV-D), that our expander maintenance protocol under any adversarially generated $\mathcal{H}$ is indeed an Υ-process. By appealing to the Sampling Lemma and Lemma 1, we show that the graph generated by using the connections prescribed by the sampled tokens have the required expansion. The upshot is that the expander maintenance protocol indeed outputs a sequence of graphs that (whp) have large expander subgraphs.

B. Expansion Analysis

To prove that a graph constructed and maintained via an Υ-process is indeed an expander, we need the following general lemma. We show that a graph in which each node has at least δ edges, each generated by picking nearly uniformly at random from a large set of the vertices, has high edge expansion α'.

Lemma 1. *Let $G = (V, E)$ be a random graph on n vertices formed by the following random process. For each vertex in V we create δ incident blue edges such that the red end of each such edge e is chosen from a probability distribution D over V defined in the following manner. For every $v \in V$, $D(v) \leqslant 1/n + 1/n^c$ for a suitably large constant c such that v has at most $\Delta - \delta$ red edges. (Hence, each node has degree at most Δ.) Furthermore, for an arbitrarily chosen subset $S_e \subseteq V$ of cardinality at least $0.8|V|$, $D(w)$ is guaranteed to be $\geqslant c_1/n$ for every $w \in S_e$, where $c_1 > 0$ is a fixed constant. Then, with high probability, G has an expansion of at least α' for some suitable constant $\alpha' > 0$.*

We need to prove that any subset S of some size c for $c \in [1, n/2]$ has sufficient expansion. To this end, we show that with probability $1/n^l$ for some constant l, the number of blue edges leaving nodes of S and pointing back to S is bounded below the minimum number needed for expansion of the set.

Proof: We now present the detailed argument. Consider a subset S of size c. We would like to upper bound the probability that too many of the blue edges incident at nodes in S have their "red" end within S itself. Let $E(U, W)$ denote the set of blue edges emanating from nodes in some set U that end at nodes in another set W. For our set S, we need to show that $|E(S, \bar{S})| \geqslant \alpha'|S|$. Since nodes in S have δ blue edges, we have $\delta|S| = |E(S, \bar{S})| + |E(S, S)|$; recall that $E(S, S)$ denotes all blue edges in S that are also red edges in S, i.e., have both endpoints in S. Thus, we require $\delta|S| - |E(S, S)| \geqslant \alpha'|S|$, and we need to upper bound $\mathbb{P}[|E(S, S)| \geqslant (\delta - \alpha')|S|]$. Let $\gamma = \delta - \alpha'$.

Let $e_1, \ldots, e_{c\delta}$ be the blue edges emanating from nodes in S. By the assumption in the statement of the lemma, we know that the endpoint for a blue edge e at node v has almost uniform probability to be any node of the set S_e, which consists of at least $0.8n$ nodes in the graph. Let X_i denote an indicator random variable which is 1 if the i-th blue edge e_i of S emanating from $u_i \in S$ has its other endpoint also inside S. It is worth pointing out that while the set S_e is different for each edge e, we assume that the set S is defined (and invariant) for each e, since we are analyzing everything at one specific time t at which all nodes of S are still in the graph. The upper bound for the probability that a blue edge leads back into S is $1/n + 1/n^c$, which we replace for simplicity in the analysis with the bound $(1 + \varepsilon)/n$, where ε is an appropriately chosen constant.

Since the probability that e points to a specific node in S is $\leqslant (1+\varepsilon)/(0.8)n$, we can sum up over all elements of S to bound the probability of e pointing to a node in S by $\leqslant (1 + \varepsilon)c/(0.8)n$. Conditioning on the event that $X_1 = 1, \ldots, X_{i-1} = 1$, decreases the number of available connection points for e inside of S and hence we have

$$\mathbb{P}[X_i = 1 \mid \bigwedge_{j=1}^{i-1} X_j = 1] \leqslant \frac{(1 + \varepsilon)c}{(0.8)n}, \tag{1}$$

for any $2 \leqslant i \leqslant c\delta$. By the chain rule of conditional probability, we have

$$\mathbb{P}[\bigwedge_{j=1}^{i} X_j = 1] = \mathbb{P}[X_i = 1 \mid \bigwedge_{j=1}^{i-1} X_j = 1] \cdot \mathbb{P}[\bigwedge_{j=1}^{i-1} X_j = 1] = \mathbb{P}[X_1 = 1] \prod_{j=2}^{i} \mathbb{P}[X_j = 1 \mid \bigwedge_{k=1}^{j-1} X_k = 1] \leqslant \left(\frac{(1 + \varepsilon)c}{(0.8)n}\right)^i, \tag{2}$$

where the upper bound follows from (1).

Now consider the following experiment where we toss m coins, and suppose that we want to know the probability that for some $a \leqslant m$, a or more coins came up heads. This probability can be upper bounded by the probability that a coins came up heads, while leaving the outcome of other coin flips unobserved. Applying this reasoning to the choices of the blue edges, we have $\binom{n}{c}$ sets of size c and thus, considering (2), we get

$$\mathbb{P}[\exists S,\ |S| = c \text{ and } |E(S, S)| \geqslant (\delta - \alpha')|S|] \leqslant \binom{n}{c}\binom{c\delta}{c\gamma}\left(\frac{(1 + \varepsilon)c}{(0.8)n}\right)^{c\gamma}.$$

We provide two separate upper bounds for the above inequality. The first upper bound handles the case for sets $c \in o(n)$. The second upper bound handles the case when $c = \kappa n$ for any positive constant $\kappa \leqslant 1/2$.

For the first case, when c is strictly sublinear, we can apply the upper bound to the binomial coefficient: $\binom{n}{k} \leqslant \left(\frac{en}{k}\right)^k$ to

obtain:

$$\begin{aligned}\binom{n}{c}\binom{\delta c}{\gamma c}\left(\frac{(1+\varepsilon)c}{0.8n}\right)^{\gamma c} &\leqslant \left(\frac{en}{c}\right)^{c}\left(\frac{e\delta c}{\gamma c}\right)^{\gamma c}\left(\frac{(1+\varepsilon)c}{0.8n}\right)^{\gamma c} = \left(\frac{e}{c}\right)^{c} c^{\gamma c}\left(\frac{(1+\varepsilon)\delta e}{\gamma 0.8}\right)^{\gamma c} n^{c(1-\gamma)} \\ &= e^{c}c^{c(\gamma-1)}n^{c(1-\gamma)}\left(\frac{(1+\varepsilon)e\delta}{\gamma 0.8}\right)^{\gamma c} = 2^{\beta' c}2^{c(\gamma-1)\log c}2^{c(1-\gamma)\log n}2^{\beta\gamma c} \\ &= 2^{c(\beta'+\beta\gamma+(\gamma-1)\log c+(1-\gamma)\log n)} \qquad (3)\end{aligned}$$

where we let $\beta = \log(((1+\varepsilon)\delta e)/(\gamma 0.8))$ and $\beta' = \log_2 e$. To show that this inequality applies, we observe that the term $(1-\gamma)\log n$ in the exponent in (3) above dominates and yields an upper bound of $2^{-l_c \log n} = n^{-l_c}$, for sufficiently large δ.

For the second case, where size $c = \kappa n$, since both n and c are very large, we can use a much tighter approximation to the binomial coefficient. We use Stirling's formula to approximate $\binom{n}{k} \sim 2^{nH(k/n)}$, where $H(\varepsilon) = -\varepsilon\log_2(\varepsilon) - (1-\varepsilon)\log_2(1-\varepsilon)$, the binary entropy of ε.

Thus when $c = \kappa n$ for κ a constant $\leqslant 1/2$, we have:

$$\binom{n}{c}\binom{\delta c}{\gamma c}\left(\frac{(1+\varepsilon)c}{0.8n}\right)^{\gamma c} \leqslant 2^{n(-\kappa\log(\kappa)-(1-\kappa)\log(1-\kappa)-\kappa\delta((\gamma/\delta)\log(\gamma/\delta)+(1-\gamma/\delta)\log(1-\gamma/\delta))+\log(\beta)\gamma\kappa)}$$

where in this case $\beta = (1+\varepsilon)c/(0.8)n$.

The expression in parentheses in the exponent will be negative for all values of κ in the specified range for sufficiently large (but still constant) κ. Therefore, we have $\mathbb{P}[\exists S,\ |S| = c \text{ and } |E(S,S)| \geqslant (\delta - \alpha')\kappa n] \leqslant 2^{-\varepsilon n}$, which will be well below n^{-l} for any constant l. Thus, over all c we pick $\max_c l_c$ and set constant δ appropriately so that $l_c \geqslant 4$. Thus, over all $\Theta(n)$ set sizes $c \in [1, n/2]$, with probability $\geqslant (1 - n^{-3})$ graph G has an edge expansion α'.

■

C. Sampling Via Random Walks

We now turn our attention to analyzing the random walks based sampling algorithm (cf. Algorithm III.1) on $\mathbf{G}$. Our goal is to leverage the result of [57] in which it is shown that random walks mix in dynamic networks without churn. Towards this goal, for a given $\mathbf{G}$, we construct another sequence of graphs $\bar{\mathbf{G}} = (\bar{\mathtt{G}}_1, \bar{\mathtt{G}}_2, \ldots)$ in which the graphs are regular and we do not have any effects of churn (cf. Algorithm IV.2). For the purpose of analysis, we simulate the random walks based sampling algorithm on $\bar{\mathbf{G}}$ so that we can apply the result from [57] (cf. Lemma 5, since $\bar{\mathtt{G}}_i$ is an expander it has a second largest eigenvalue bounded away from 1), and show how to translate the result back to $\mathbf{G}$ in a suitable manner (cf. Lemma 6). We are now able to make the following claims:

1: // We describe how each $\bar{\mathtt{G}}_i = (\bar{\mathtt{V}}_i, \bar{\mathtt{E}}_i)$ is constructed at the start of round i based on $\mathtt{G}_i$ created in line number 23 of the Υ-process.
2: // Assume that $\bar{\mathtt{G}}_1 = \mathtt{G}_1$ and for all $i > 1$, $\bar{\mathtt{G}}_{i-1}$ has been constructed. So we will focus on constructing $\bar{\mathtt{G}}_i$.
3: $\bar{\mathtt{V}}_i = \mathtt{V}_i$.
4: $\bar{\mathtt{E}}_i = \mathtt{E}_i$. Designate edges in $\bar{\mathtt{E}}_i$ to be real edges.
5: Consider any arbitrary bijection B between $\bar{\mathtt{V}}_i \setminus \bar{\mathtt{V}}_{i-1}$ and $\bar{\mathtt{V}}_{i-1} \setminus \bar{\mathtt{V}}_i$, which exists because $|\bar{\mathtt{V}}_i \setminus \bar{\mathtt{V}}_{i-1}| = |\bar{\mathtt{V}}_{i-1} \setminus \bar{\mathtt{V}}_i|$.
6: **for** each $v \in \bar{\mathtt{V}}_i \setminus \bar{\mathtt{V}}_{i-1}$ **do**
7: Copy the state of $B(v) \in \bar{\mathtt{V}}_{i-1} \setminus \bar{\mathtt{V}}_i$ on to v. In particular, v now has all the random walk tokens previously in $B(v)$. // Thus, in effect, there is no churn.
8: Tag all vertices in $\bar{\mathtt{V}}_i$ that have no self-loops.
9: Ensure that vertices with self-loops are untagged. The number of untagged vertices is $> 0.8n$ because $\delta < \Delta/6$, so the at most $n\delta$ blue edges would have exhausted red edges in fewer than $0.2n$ nodes.
10: **for** each dangling port p in some $v \in \bar{\mathtt{V}}_i$ **do**
11: Pick a vertex v' uniformly at random from a set of untagged vertices.
12: Pick a self-loop port p' in v' and tag v' so it is not picked again.
13: Add an edge $e = (v, v')$ to $\bar{\mathtt{E}}_i$ connecting p to p'.
14: Designate e to be a *ghost* edge.
15: Finalize $\bar{\mathtt{G}}_i = (\bar{\mathtt{V}}_i, \bar{\mathtt{E}}_i)$.

Algorithm IV.2: Procedure to construct $\bar{\mathbf{G}}$

Lemma 2. *For all i, the graph $\bar{\mathtt{G}}_i$ has expansion α' for some constant $\alpha' > 0$, with high probability. In addition, for each $\mathtt{G}_i$, there exists a subgraph $\hat{H}_i$ of size $n - o(n)$ with constant expansion α for $\alpha' \geqslant \alpha > 0$.*

The result is established by showing that Lemma 1 applies to $\bar{\mathtt{G}}_i$ for the first statement. For the second statement, we use a modification of a result found in [58] and show that by removing all nodes incident to a ghost edge in $\bar{\mathtt{G}}_i$ we end up with a subgraph that has expansion α. This subgraph is isomorphic to a subgraph of $\mathtt{G}_i$, which therefore also has expansion α.

Proof of Lemma 2: For showing that $\bar{\mathtt{G}}_i$ has sufficient expansion, we will show that the preconditions for Lemma 1 hold. The edge set $\bar{\mathtt{E}}_i$ consists of real edges and ghost edges as described above. Considering the union of both sets of edges, every node $v \in \bar{\mathtt{V}}_i$ has δ blue edges, and the endpoint of each blue edge is chosen from a set V' of untagged vertices, which will have cardinality at least $0.8n$. This can be seen by using a simple balls-in-bins argument. If one tosses δn balls into bins with capacity at least $(\Delta - \delta)/2$, for appropriately chosen Δ and δ, one can ensure that at least $0.8n$ of the bins are not at capacity. Note that the actual bin capacity is $\Delta - \delta$, but we cut it in half artificially for the proof to ensure sufficient space for new blue edges. Since this capacity condition is constantly maintained over the evolution of the graphs $\bar{\mathtt{G}}_i$, for each new blue edge formed, there will always be at least $0.8n$ nodes with at least $(\Delta - \delta)/2$ red edges with self loops to choose from. Furthermore, for every v, for every edge e with blue endpoint at v, by the construction above the red end of e is chosen uniformly at random from set V'. Therefore, we can apply Lemma 1 and it follows that $\bar{\mathtt{G}}_i$ is an expander with expansion α' for some constant α' with high probability.

For the second part of the statement in the Lemma, we can apply an adaptation of a result in [58], which states that a graph G with expansion ϕ can have the edges of an arbitrary fraction ε of the nodes adversarially removed and yet still contain a subgraph of size $\Theta(n)$ that has a slightly decreased (but still constant) expansion. For our setting, we need a variant of this result that the remaining subgraph has size $\geqslant n - o(n)$, if a set of size $o(n)$ is removed from the original graph (cf. Lemma 3). For our graph $\bar{\mathtt{G}}_i$, we look at the subgraph induced by all nodes that are not incident to one or more ghost edges. We note that this subgraph, $\hat{H}_i$ is identical in both $\bar{\mathtt{G}}_i$ and $\mathtt{G}_i$ (i.e. there is a one-to-one mapping of nodes and edges from $\hat{H}_i \subset \bar{\mathtt{G}}_i$ to a subgraph $\hat{H}_i' \subset \mathtt{G}_i$). Applying Lemma 3, establishes the second claim in the lemma. ∎

For completeness, we restate the following lemma; the original proof is in [59].

Lemma 3. *Let G be a n-node graph with expansion ϕ and constant node degree d and suppose that all nodes in a set F are removed from G, where $|F| = o(n)$. Then, for any positive constant $c < 1$, there exists a subgraph H of G such that*

1) *H has expansion $c\phi$, and*
2) *$|H| \geqslant n - |F|\left(1 + \frac{1}{\phi(1-c)}\right)$.*

Proof: We adapt the proof of Theorem 2.1 in [58]. Let G_F be the graph yielded by removing the set F from G. Perform the following iterative procedure (cf. Algorithm Prune in [58]): Initialize $G_0 = G_F$. In iteration $i \geqslant 0$, let S_i be any set of up to $|V(G_i)|/2$ nodes with expansion smaller than $c\phi$. If S_i exists, prune S_i from G_i, i.e., $G_{i+1} = G_i \setminus S_i$. Let H be graph that we get after the final iteration; note that H has expansion $\geqslant c\phi$.

We now lower bound the size of H. Suppose that the pruning procedure stops after m iterations. For the sake of a contradiction, suppose that

$$\frac{|F|}{\phi(1-c)} < \left|\bigcup_{i=0}^{m} S_i\right|.$$

Define the set $S = \bigcup_{i=0}^{\ell} S_i$ where ℓ is the smallest index such that exactly one of the following two cases holds:

First, assume that $|S| \leqslant n/2$. Let $N_{G_i}(S_j)$ denote the neighbors in $G_i \setminus S_j$ of nodes in set S_j. We make use of the following result whose proof follows analogously to Lemma 2.6 of [58]:

Lemma 4 (cf. Lemma 2.6 in [58])**.** *Suppose that we execute procedure* Prune(c)*. For all j with $0 \leqslant j < m$, it holds that $|N_{G_F}\left(\bigcup_{i=0}^{j} S_i\right)| \leqslant c\phi|\bigcup_{i=0}^{j} S_i|$.*

Since G has expansion of ϕ, it holds that $|N_G(S)| \geqslant \phi|S|$. On the other hand, Lemma 4 implies that $|N_{G_F}(S)| \leqslant c\phi|S|$. Thus we get $|F| \geqslant \phi|S| - c\phi|S| = \phi(1-c)|S|$ and hence $|S| \leqslant |F|/(\phi(1-c))$, yielding a contradiction.

Now, consider the case where $|S| > n/2$. Then, it follows that

$$\left|\bigcup_{i=0}^{\ell-1} S_i\right| \leqslant \frac{|F|}{\phi(1-c)}, \tag{4}$$

but $|S_\ell| > n/2 - \frac{|F|}{\phi(1-c)}$. (Note that if $S_\ell \leqslant n/2 - \frac{|F|}{\phi(1-c)}$, then $|S| \leqslant n/2$ and the first case applies.) Recalling that $F = o(n)$, it follows that $S_\ell \in \Theta(n)$. Since S_ℓ was removed when executing Prune(c), we know that $|N_{G_\ell}(S_\ell)| \leqslant c\phi|S_\ell|$ and, by the

expansion of G, we have $|N_G(S_\ell)| \geqslant \phi|S_\ell|$ as $S_\ell \leqslant n/2$. Thus

$$|N_G(S_\ell)| - |N_{G_\ell}(S_\ell)| \geqslant \phi(1-c)|S_\ell| = \Theta(n)$$

We observe that the size of the neighborhood of S_ℓ must have been reduced by $\Theta(n)$ either due to the removal of nodes in F or because of the pruning of the sets $S_0, \ldots, S_{\ell-1}$. This, however, yields a contradiction, since $|F| + \bigcup_{i=0}^{\ell-1} S_i = O(|F|) = o(n)$.

Thus we have shown that $|\bigcup_{i=0}^{m} S_i| \leqslant \frac{|F|}{\phi(1-c)}$ and hence $|H| \geqslant |G| - |F|(1 + 1/(\phi(1-c))$, which completes the proof. ■

Now consider the execution of the sampling algorithm (cf. Algorithm III.1) on $\bar{\mathbf{G}}$ (instead of $\mathbf{G}$) in which any dropped token (that chose a dangling port p in $\mathbf{G}$) is actually sent along the ghost edge (incident at p). Once a token goes through a ghost edge, it is called a ghost token. Tokens that never passed along ghost edges are called real tokens. Therefore, we can apply the results from [57] to get the following lemma.

Lemma 5 (cf. [57])**.** *For the dynamic graph $\bar{\mathbf{G}} = (\bar{\mathtt{G}}_i)_{i \geqslant 1}$ described above, there is a $\tau \in \Theta(\log n)$ that we call its mixing time such that, if a node received a random walk token t in round i that originated in round $i' = i - \tau$, then, the probability that t originated from any particular node in $\bar{\mathtt{V}}_{i'}$ is within $[1/n - 1/n^c, 1/n + 1/n^c]$ for any fixed c that depends on the constant hidden in the asymptotic description of τ.*

The definition of a mature token (cf. Section III) is based on the parameter τ defined in Lemma 5. It must be noted that in the statement of Lemma 5, we are precluding any knowledge of whether t is a ghost token or a real token. If, for instance, we know a posteriori that t is a real token, but most of the tokens received by the node v were ghost tokens, then, Lemma 5 cannot be applied. Under this scenario, the distribution of starting vertices for t is not guaranteed to be uniform. However, if a node receives a sufficiently large number of real tokens, then, intuitively, we can conclude that that node is well connected and likely to receive well-mixed random walk tokens. We now wish to formalize this observation.

The following Sampling Lemma shows the key property, namely the near-uniform distribution of tokens in the Υ process graph $\mathtt{G}_i$, by appealing to the distribution of real tokens (not ghost tokens) in $\bar{\mathtt{G}}_i$.

Lemma 6 (Sampling Lemma)**.** *Recall that z is the number of random walk tokens initiated by each node every round. Let d be any node in $\bar{\mathtt{G}}_i$ (at round i) and let T_d be the set of mature real tokens received by d in round i.*

1) *The number of nodes $|\{d \in \bar{\mathtt{V}}_i \mid T_d \geqslant (1-\eta)z\}|$ that receive at least $(1-\eta)z$ mature tokens is (whp) at least $n - O(n/\log^k n)$ when $z \in \Omega(\log^{2k+1} n)$.*
2) *Furthermore, if $|T_d| \geqslant (1-\eta)z$, then we are guaranteed with high probability that for each $t \in T_d$, there is a set $S \subseteq \bar{\mathtt{V}}_{i-\tau}$ of cardinality at least $(1-\varepsilon)n$, for some small $\varepsilon > 0$, such that the probability that t originated at any $s \in S$ is in $\Theta(1/n)$. (We note specifically, that the upper bound on the probability is at most $1/n + 1/n^c$ for some suitably large constant c.)*

Proof: Consider the following experiment. Let t be a token chosen uniformly at random from the set of all tokens that were initiated at round $i - \tau$. Let R_d be the event that the token t reached d at time step i as a real token, i.e., $t \in T_d$. (The arrival at time step i implies that t matured on entering d.) As a consequence of Lemma 5, we get $\mathbb{P}[R_d] \leqslant 1/n + 1/n^c$ for any fixed c, but we cannot lower bound $\mathbb{P}[R_d]$ using Lemma 5 because R_d requires the token t to reach d as a real token, which will not happen (for example) when d is connected only via ghost edges.

Consider some $d \in \bar{\mathtt{V}}_i$ such that $\mathbb{P}[R_d] \leqslant c'/n$ for any fixed c' that is less than $1 - \eta$. Then, $\mathrm{E}[|T_d|] \leqslant c'z$; recall that t is a random token initiated in round $i - \tau$. Under this circumstance, $\mathbb{P}[|T_d| \geqslant (1-\eta)z] \leqslant 1/\mathrm{poly}(n)$, as long as z is sufficiently large. (This follows by an application of a Chernoff bound which is possible due to the independence of the random walks. In particular, whether a random walk reaches d or not is independent of other random walks.) Thus, we condition the rest of the proof on $T_d < (1-\eta)z$ for all d with $\mathbb{P}[R_d] \leqslant c'/n$. Under this conditioning, if $|T_d| \geqslant (1-\eta)z$, then $\mathbb{P}[R_d] > c'/n$ (which we will use at the end of the proof).

To focus our attention on such nodes that receive at least $(1-\eta)z$ tokens, we define $D := \{d \in \bar{\mathtt{V}}_i || T_d| \geqslant (1-\eta)z\}$. We now wish to prove that $|D|$ is large (i.e., item (2) of the lemma). We do this by showing that (i) whp no node in D gets too many tokens and that (ii) there are sufficiently large number of real tokens, thus implying that D must be large enough to ensure that all the real tokens have recipient nodes in D. To show (i), consider any $d \in D$; it has $\mathrm{E}[|T_d|] \leqslant (1 + 1/n^{c-1})z$. A simple application of Chernoff bound ensures that the $\mathbb{P}[|T_d| \geqslant (1 + 1/\log^k n)z] \leqslant 1/\mathrm{poly}(n)$, when $z \in \Omega(\log^{2k+1} n)$. We now need to prove that $\sum_d |T_d|$ is large. Notice that $\bar{\mathbf{G}}_i$ are Δ-regular and the number of ghost edges is at most $O(n/\log^{k-1} n)$ and whp the number of tokens passing through a ghost edge is at most $O(z)$. Therefore, whp, the number of tokens that become ghost tokens (over all ghost edges in all $\tau \in O(\log n)$ rounds) is at most $O(nz/\log^{k-2} n)$. Thus, at least $nz - O(nz/\log^{k-2} n)$ tokens are (whp) real. With items (i) and (ii) (mentioned earlier in this paragraph) proved, one can deduce that $|D| \geqslant n - O(n/\log^k n)$ (whp), when $z \in \Omega(\log^{2k+1} n)$. This proves the first statement in the lemma.

Consider a random walk token t chosen uniformly at random from the set of all tokens that originated in round $i-\tau$. To prove the second statement, let O_s denote the event that a random walk token t originated at $s \in \bar{\mathtt{V}}_{i-\tau}$ in round $i-\tau$. Thus, $\mathbb{P}[O_s] = 1/n$. Now fix a $d \in D$ and define a corresponding $S = \{s \in \bar{\mathtt{V}}_{i-\tau} | \mathbb{P}[R_d|O_s] > c''/n\}$ for any fixed c'', $0 < c'' < c'$. We want to show that S is large, i.e., of size at least $(1-\varepsilon)n$. Recall that $E[\text{number of real tokens that reached } d] = z\sum_s \mathbb{P}[R_d|O_s] \geqslant c'z$ (since $\mathbb{P}[R_d] \geqslant c'/n$). Thus, we can conclude that

$$\sum_s \mathbb{P}[R_d|O_s] \geqslant c'. \tag{5}$$

Recall that $\mathbb{P}[R_d|O_s] \leqslant 1/n + 1/n^c \leqslant (1+\varepsilon')/n$ for every s and any $\varepsilon' > 0$. Let $K = n - |S|$. To see how large K can get, we set $|S|$ summation terms in Equation (5) to $(1+\varepsilon')/n$ and the remaining K terms to c''/n. Solving for K, we get $K < n\left(\frac{1+\varepsilon'-c'}{1+\varepsilon'-c''}\right)$. Recall that c' can be any fixed constant strictly less than 1 while c'' can be any fixed constant strictly larger than 0. Thus, we can make $\varepsilon = \left(\frac{1+\varepsilon'-c'}{1+\varepsilon'-c''}\right)$ arbitrarily small and thereby leading to $|S| \geqslant (1-\varepsilon)n$ for any fixed $\varepsilon > 0$.

Now fix a $d \in D$ and an $s \in S$ (where S is chosen corresponding to the d). Let t be chosen uniformly at random from the set of all tokens that originated in round $i-\tau$. Since $\mathbb{P}[O_s|R_d] = \frac{\mathbb{P}[R_d|O_s]\mathbb{P}[O_s]}{\mathbb{P}[R_d]}$ and all the terms in the right hand side are $\Theta(1/n)$, it follows that $\mathbb{P}[O_s|R_d] \in \Theta(1/n)$ for all $s \in S$, thus proving item (2) of the lemma. ■

D. Viewing the Expander Maintenance Protocol as an Υ-Process

We start by arguing that at some round $i \in \Theta(\log n)$, G_i is going to be an expander. This will form the basis upon which our inductive statement (cf. Lemma 7) hinges. We note that the bootstrap phase runs for $B = \beta \log n$, where the constant β is sufficiently large and therefore, the bootstrap phase (along with the safety of a steady stream of mature tokens that starts after the first τ rounds) lasts sufficiently long for the network to form the first expander. This formation of the first expander can be easily argued based on (i) Lemma 5, which proves that the nodes will get a steady stream of mature tokens after the first τ rounds, and (ii) Lemma 1, which ensures that, once all the nodes connect using blue edges (which will happen in at most $B = \Theta(\log n)$ rounds, where B is the length of the bootstrap phase), the graph is indeed an expander (whp).

In the proof of the following lemma we make use of the fact that, when a node u initiates a blue edge to some node v in the expander maintenance protocol, then v was (almost) uniformly chosen from a large set, as described in the Υ-process (cf. Algorithm IV.1).

In light of Lemma 2, it is sufficient to show that, if the expander maintenance protocol (in conjunction with the adversary's behavior) has been an Υ-process until some round $i-1$ (for $i \geqslant \Theta(\log n)$), then it will continue to be an Υ-process in round i (whp); we will formalize this now.

Lemma 7. *Suppose the expander maintenance protocol (cf. Section III) has been an Υ-process until round $i-1$. Then, with high probability, the expander maintenance protocol will continue to be an Υ-process in round i and (as a consequence of Lemma 2) the graph G_i that it creates at time step i will also contain a large $n - o(n)$-sized expander subgraphs with expansion at least α.*

Proof: To see that this is indeed the case, we walk through the activities of both an Υ-process (as required by its definition), and the expander maintenance protocol.

As argued above, the graph after the end of the bootstrap phase, i.e., G_{B+1}, is an expander ($B = \Theta(\log n)$). The Υ-process also starts with the graph being an expander (line 1 of Algorithm IV.1). We next show that the graph generated henceforth by the protocol (i.e., G_{B+2} onwards) will be an Υ-process.

It is easy to see that until line number 7, the behavior of the Υ-process matches the behavior of the adversary.

Recall that in the expander maintenance protocol, when a node has fewer than δ blue edges, it tries to connect until it succeeds (or is churned out). Thus, it remains deficient of blue edges until some round and then gets an edge chosen uniformly at random. The Υ-process delineates the two decisions, namely, (i) when to add an edge and (ii) how to add an edge.

From line 8 until line number 15, the Υ-process is required to ensure two conditions listed under line number 9. These lines do not actually create the edges, but rather place budgets on the number of blue edges that need to be added to each node. After the budgets are set, the edges are actually created at a later part (from line number 16 to 20) of the Υ-process definition. We need to prove that (whp) the expander maintenance protocol will also maintain the two conditions and we will revisit this shortly.

The edges are actually added between line number 16 to 20 of the Υ-process definition, where the red ends are chosen almost uniformly at random from a large set of vertices. Since the RW-Sampling algorithm has been successfully running up until round $i-1$ (as a consequence of our assumption that the expander maintenance protocol has been an Υ-process until

time step $i-1$ in conjunction with Lemma 6), the expander maintenance protocol also chooses the red end from the current set of vertices based on a distribution that is almost uniform over a large fraction of current (i.e., round i) vertices.

In the **For** loop starting in line number 10, the Υ-process refreshes the edges of vertices chosen at random, which the expander maintenance protocol also performs. After the decision to refresh has been made, the edges (as usual) are chosen almost uniformly at random from a large fraction of current nodes.

We now have to prove the two conditions (see line number 9, conditions (2) and (3)) that any Υ-process must ensure are in fact upheld by the expander maintenance protocol. Denote by X_i the nodes that start to have fewer than δ blue edges in round i, i.e., they either had all δ blue edges in round $i-1$ or were only churned in at the current round i. In particular, we focus on the set $X_{i-\Theta(\log n)}$. In order to show that the two conditions are upheld, it suffices to show the following claim.

Claim 1. *The nodes in $X_{i-\Theta(\log n)}$ have all obtained the requisite number of blue edges at least once since round $i-\Theta(\log n)$ w.h.p.*

Before proving the claim, we first see why it is sufficient. Since in any round at most $O(n/\log^k n)$ nodes can become deficient of blue edges (i.e., fewer than δ blue edges) and all of them reconnect w.h.p within $O(\log n)$ rounds (as a consequence of the claim), the number of nodes that are deficient of blue edges is at most $O(n/\log^{k-1} n)$, and thus both conditions will be ensured.

Proof of Claim 1: Consider a node $f \in X_{i-\Theta(\log n)}$ that received a fresh set of $\Omega(\log^2 n)$ mixed tokens in round $i-\Theta(\log n)$. The node f retains some of its tokens to itself and the remaining set T of tokens (that number $O(\log^2 n)$) are given to any node(s), say node a, that might be attached to f by the adversary. Now suppose a has still not received any fresh tokens and another node b is attached on to it. Then, the same set T of tokens are passed along by a to b. In fact, a constant number of such new nodes may be attached to a, and all of them will share the same set T of tokens. Thus, let $Y_T(j)$ be the set of nodes in round $i-\Theta(\log n)+j$, $0 \leqslant j \leqslant \Theta(\log n)$, that are in possession of the same set T of tokens and are trying to get fresh tokens. Note that $Y_T(0)$ is a constant. As long as $|Y_T(j)|$ stays bounded within $O(\log n)$, the probability that a node in $Y_T(j)$ does not get fresh tokens in round $i-\Theta(\log n)+j$ is conservatively bounded from above by $1/\log n$. This is because, the node can fail to get fresh tokens because (i) it made requests for fresh tokens to nodes that were already requested for fresh tokens by other nodes in $Y_T(j)$, (ii) it made requests for fresh tokens to nodes that were already requested for fresh tokens by nodes outside of $Y_T(j)$ (but such requests are made by nodes with a token set independent of T and there can only be $O(n/\text{polylog}n)$ such requests), or (iii) it made requests to nodes that did not themselves possess fresh tokens (which can only be $O(n/\text{polylog}n$ many in number). Thus, $E[Y_T(j+1)] \leqslant \varepsilon E[Y_T(j)]$ for any $\varepsilon > 1/\log n$. Recursively, $E[Y_T(\Theta(\log n))] \in \varepsilon^{\Theta(\log n)} E[Y_T(0)] \leqslant 1/n^c$, for any constant c. Thus, by a straightforward application of the Markov's inequality, we can ensure that $\mathbb{P}[Y_T(\Theta(\log n)) \geqslant 1] \leqslant 1/n^c$. The same argument can also be made for nodes possessing fresh tokens and trying to gain the required δ blue edges. Moreover, the same argument can be made over the entire set $X_{i-\Theta(\log n)}$. Each node can fail to get its requisite fresh tokens or the requisite δ blue edges (whichever the case may be) with probability at most $O(1/\log n)$. Thus, we again see that the expected number of nodes in $X_{i-\Theta(\log n)}$ that have not yet acquired their requisite number of blue edges dwindles exponentially, and a similar Markov's inequality application completes the argument. Thus, by the current round i, the nodes in $X_{i-\Theta(\log n)}$ have all obtained the requisite number of blue edges at least once since round $i-\Theta(\log n)$ (whp), thus completing the proof of Claim 1 and Lemma 7. ■

■

From the above, our main theorem follows:

Theorem 1. *The expander maintenance protocol maintains the network communication graph sequence $G_1, G_2, \ldots,$ as graphs that contain large $n-o(n)$-sized expander graphs with expansion at least α for a number of rounds that is at least a polynomial in n with high probability.*

V. Impossibility of Maintaining Connectivity in the Classic Synchronous Round Model

Recall that, during a single round in our model (cf. Section II), a node u can send a message to a node v, which can then perform some local computation (taking into account the message received from u), and finally v can send some messages that are delivered by the end of the round. We assume that this entire sequence is atomic with respect to the actions of the adversary, i.e., nodes do not join or leave the network before the sequence is complete. This is in contrast to the classic synchronous model where such a sequence would span 2 rounds, because any message sent in round r can only be processed by the receiver at the start of round $r+1$. While it might appear that this assumption is just a mere technicality, we will now prove that there is no algorithm that can maintain connectivity (let alone expansion) for a majority of the nodes in the classic synchronous round model, even if the churn is limited to only $\Theta(\sqrt{n\,\text{polylog}(n)})$ nodes per round.

Theorem 2 (Impossibility without Single-Round Handshakes)**.** *Let m and c be arbitrary positive constants such that $c \geqslant m \geqslant 1$. Consider a synchronous n-node network where an oblivious adversary can subject $C(n) = \Omega(\sqrt{n \log^c n})$ nodes to churn in each round. Moreover, suppose that every node can send messages of size $O(\log^m n)$ to at most Δ distinct nodes per round, where $\Delta \geqslant 2$ is an arbitrary fixed integer. Then, there is an adversarial strategy that, with high probability, leads to a round where the network contains at least $n/2 - o(n)$ nodes that are partitioned into $\Theta(n)$ (disconnected) components.*

We present a high-level overview of the proof. The main idea is to add new nodes in layers of exponentially increasing size while limiting the information flow between these newly added nodes and the remaining network. This ensures that, after the construction of the last layer (containing $C(n)$ nodes), only a polylogarithmic number of the IDs of nodes in this layer are known to the remaining network and vice versa. Moreover, we need to ensure that the nodes in the layer do not know too much about each other to avoid the case where the layer itself forms a well-connected graph. We leverage the low information flow to create a bottleneck that slows down the integration of these newly added nodes into the remaining network. Note that, up until the point where the layer nodes can communicate successfully with the remaining network, they will form small disconnected components. We then repeat this layer construction process several times at other points in the network, which results in a situation where at least half of the nodes are in such layers (and hence disconnected). For lack of space, we defer the full proof to the full version of the paper.

VI. Conclusion

We presented a protocol that guarantees that the underlying topology has high expansion despite large amount of dynamism and adversarial churn. Since high expansion is needed for implementing many fundamental distributed computing tasks in an efficient and robust manner, this protocol serves as a basic ingredient. It will be interesting to improve our protocol to tolerate even higher churn (say, linear in n) and different types of adversaries.

Acknowledgements

John Augustine, Gopal Pandurangan, and Scott Roche are thankful to the Institute for Computational and Experimental Research in Mathematics (ICERM), Brown University, Providence, RI 02906 for hosting them for a semester. Part of this work was done while the above three authors were at the ICERM.

References

[1] Crashplan Inc., "http://www.crashplan.com/."

[2] Symform:, "http://www.symform.com/."

[3] A. Peddemors, "Cloud storage and peer-to-peer storage - end-user considerations and product overview", 2010. *https://www.dropbox.com/s/af4f7xx0rpnn95o/152.pdf?dl=0*, 2010.

[4] J. Falkner, M. Piatek, J. P. John, A. Krishnamurthy, and T. E. Anderson, "Profiling a million user DHT," in *IMC*, 2007, pp. 129–134.

[5] P. K. Gummadi, S. Saroiu, and S. D. Gribble, "A measurement study of napster and gnutella as examples of peer-to-peer file sharing systems," *Computer Communication Review*, vol. 32, no. 1, p. 82, 2002.

[6] S. Sen and J. Wang, "Analyzing peer-to-peer traffic across large networks," in *IMC*, 2002, pp. 137–150.

[7] D. Stutzbach and R. Rejaie, "Understanding churn in peer-to-peer networks," in *IMC*, 2006, pp. 189–202.

[8] P. Druschel and A. Rowstron, "Storage management and caching in past, a large-scale, persistent peer-to-peer storage utility," *SOSP*, 2001.

[9] P. Druschel and A. Rowstron, "Past: A large-scale, persistent peer-to-peer storage utility," in *HotOS VIII*, 2001, pp. 75–80.

[10] Y.-W. Chan, T.-H. Ho, P.-C. Shih, and Y.-C. Chung, "Malugo: A peer-to-peer storage system," *Int. J. ad hoc and ubiquitous computing*, vol. 5, no. 4, 2010.

[11] R. Hasan, Z. Anwar, W. Yurcik, L. Brumbaugh, and R. Campbell, "A survey of peer-to-peer storage techniques for distributed file systems," in *Proc of ITCC*, pp. 205–213.

[12] J. F. Canny, "Collaborative filtering with privacy," in *IEEE Symposium on Security and Privacy*, 2002, pp. 45–57.

[13] Cloudmark Inc., "http://cloudmark.com/."

[14] S. Datta, K. Bhaduri, C. Giannella, R. Wolff, and H. Kargupta, "Distributed data mining in peer-to-peer networks," *IEEE Internet Computing*, vol. 10, no. 4, pp. 18–26, 2006.

[15] D. J. Malan and M. D. Smith, "Host-based detection of worms through peer-to-peer cooperation." in *WORM*, V. Atluri and A. D. Keromytis, Eds. ACM Press, 2005, pp. 72–80.

[16] V. Vlachos, S. Androutsellis-Theotokis, and D. Spinellis, "Security applications of peer-to-peer networks," *Comput. Netw.*, vol. 45, pp. 195–205, June 2004.

[17] R. Geambasu, T. Kohno, A. A. Levy, and H. M. Levy, "Vanish: Increasing data privacy with self-destructing data," in *USENIX Security Symposium*, 2009, pp. 299–316.

[18] O. Babaoglu, M. Merzolla, and M. Tamburini, "Design and implementation of a P2P cloud system," in *SAC*, 2012. pp. 412-417.

[19] J. Augustine, G. Pandurangan, P. Robinson, and E. Upfal, "Towards robust and efficient computation in dynamic peer-to-peer networks," in *SODA*, 2012, pp. 551–569.

[20] C. Dwork, D. Peleg, N. Pippenger, and E. Upfal, "Fault tolerance in networks of bounded degree," *SIAM J. Comput.*, vol. 17, no. 5, pp. 975–988, 1988.

[21] J. Augustine, G. Pandurangan, and P. Robinson, "Fast byzantine agreement in dynamic networks," in *PODC*, 2013, pp. 74–83.

[22] J. Augustine, A. R. Molla, E. Morsy, G. Pandurangan, P. Robinson, and E. Upfal, "Storage and search in dynamic peer-to-peer networks," in *SPAA*, 2013, pp. 53–62.

[23] G. Pandurangan, P. Robinson, and A. Trehan, "Dex: Self-healing expanders," in *IPDPS*, 2014, pp. 702-711.

[24] C. Law and K.-Y. Siu, "Distributed construction of random expander networks," in *INFOCOM*, 2003, pp. 2133 – 2143.

[25] G. Pandurangan, P. Raghavan, and E. Upfal, "Building low-diameter P2P networks," in *FOCS*, 2001, pp. 492–499.

[26] Y. Afek, B. Awerbuch, and E. Gafni, "Applying static network protocols to dynamic networks," in *FOCS*, 1987, pp. 358–370.

[27] S. Dolev, *Self-stabilization*. Cambridge, MA, USA: MIT Press, 2000.

[28] E. Gafni and B. Bertsekas, "Distributed algorithms for generating loop-free routes in networks with frequently changing topology," *IEEE Trans. Comm.*, vol. 29, no. 1, pp. 11–18, 1981.

[29] Y. Afek, E. Gafni, and A. Rosen, "The slide mechanism with applications in dynamic networks," in *PODC*, 1992, pp. 35–46.

[30] B. Awerbuch, B. Patt-Shamir, D. Peleg, and M. E. Saks, "Adapting to asynchronous dynamic networks," in *STOC*, 1992, pp. 557–570.

[31] B. Awerbuch and M. Sipser, "Dynamic networks are as fast as static networks (preliminary version)," in *FOCS*, 1988, pp. 206–220.

[32] T. Jacobs and G. Pandurangan, "Stochastic analysis of a churn-tolerant structured peer-to-peer scheme," *Peer-to-Peer Networking and Applications*, 6(1): 1-14 (2013).

[33] R. Guerraoui, F. Huc, and A. Kermarrec, "Highly dynamic distributed computing with byzantine failures," in *PODC*, 2013, pp. 176–183.

[34] F. Kuhn, S. Schmid, and R. Wattenhofer, "Towards worst-case churn resistant peer-to-peer systems," *Distributed Computing*, vol. 22, no. 4, pp. 249–267, 2010.

[35] R. Jacob, A. W. Richa, C. Scheideler, S. Schmid, and H. Täubig, "A distributed polylogarithmic time algorithm for self-stabilizing skip graphs," in *PODC*, 2009, pp. 131–140.

[36] G. Pandurangan and A. Trehan, "Xheal: localized self-healing using expanders," in *PODC*, 2011, pp. 301–310.

[37] B. Awerbuch and C. Scheideler, "The hyperring: a low-congestion deterministic data structure for distributed environments," in *SODA*, 2004, pp. 318–327.

[38] P. Mahlmann and C. Schindelhauer, "Peer-to-peer networks based on random transformations of connected regular undirected graphs," in *SPAA*, 2005, pp. 155–164.

[39] A. Fiat and J. Saia, "Censorship resistant peer-to-peer content addressable networks," in *SODA*, 2002, pp. 94–103.

[40] K. Hildrum and J. Kubiatowicz, "Asymptotically efficient approaches to fault-tolerance in peer-to-peer networks." in *DISC*, 2003, pp. 321–336.

[41] M. Naor and U. Wieder, "A simple fault tolerant distributed hash table," in *IPTPS*, 2003, pp. 88–97.

[42] C. Scheideler, "How to spread adversarial nodes?: rotate!" in *STOC*, 2005, pp. 704–713.

[43] B. Awerbuch and C. Scheideler, "Towards a scalable and robust DHT," *Theory of Computing Systems*, vol. 45, pp. 234–260, 2009.

[44] V. King, J. Saia, V. Sanwalani, and E. Vee, "Towards secure and scalable computation in peer-to-peer networks," in *FOCS*, 2006, pp. 87–98.

[45] R. O'Dell and R. Wattenhofer, "Information dissemination in highly dynamic graphs," in *DIALM-POMC*, 2005, pp. 104–110.

[46] F. Kuhn and R. Oshman, "Dynamic networks: Models and algorithms," *SIGACT News*, vol. 42(1), pp. 82–96, 2011.

[47] F. Kuhn, N. Lynch, and R. Oshman, "Distributed computation in dynamic networks," in *ACM STOC*, 2010, pp. 513–522.

[48] C. Scheideler and S. Schmid, "A distributed and oblivious heap," in *ICALP*, 2009, pp. 571–582.

[49] A. Rowstron and P. Druschel, "Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems," in *Proc. of the IFIP/ACM Intenrational Conference on Distributed Systems Platforms*, 2001, pp. 329–350.

[50] B. Zhao, J. Kubiatowicz, and A. Joseph, "Tapestry: An infrastructure for fault-tolerant wide-area location and routing," *Technical Report UCB/CSD-01-1141, UC Berkeley*, April, 2001.

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

[52] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker, "A scalable content addressable network," in *SIGCOMM*, 2001, pp 161-172.

[53] M. Kashoek and D. Karger, "Koorde: A simple degree optimal distributed hash table," in *In IPTPS*, 2003, pp. 98-107.

[54] M. Naor and U. Wieder, "Scalable and dynamic quorum systems," in *PODC*, 2003, pp. 114-122.

[55] U. Nadav, and M. Naor, "Scalable and dynamic quorum system", *Distributed Computing*, 17(4), pp. 311-322, 2005.

[56] D. Peleg, *Distributed Computing: A Locality-Sensitive Approach*, SIAM, 2000.

[57] A. D. Sarma, A. R. Molla, and G. Pandurangan, "Distributed computation in dynamic networks via random walks," *Theoretical Computer Science*, 581, 45-66, 2015. Conference version: *DISC*, 2012, pp. 136–150.

[58] A. Bagchi, A. Bhargava, A. Chaudhary, D. Eppstein, and C. Scheideler, "The effect of faults on network expansion," *Theory Comput. Syst.*, vol. 39, no. 6, pp. 903–928, 2006.

[59] J. Augustine, G. Pandurangan, and P. Robinson, "Fast byzantine leader election in dynamic networks," in *DISC*, 2015, accepted for publication.