Distributed Algorithmic Foundations of Dynamic Networks

John Augustine[*] Gopal Pandurangan[†] Peter Robinson[‡]

1 Introduction and Motivation

Dynamic Networks. Large real-world networks are inherently very dynamic: the participants in peer-to-peer (P2P) networks change over time, mobile nodes in wireless networks move in and out of each other's transmission range, and, in distributed data center networks, faulty machines need to be replaced by new machines without interrupting the operation of the remaining network. Furthermore, they are resource-constrained, unreliable, and vulnerable to attacks. For example, in P2P networks, peers (nodes) join and leave at a high rate and the underlying network topology also changes continuously over time; these networks are bandwidth-constrained, unreliable due to the high node turnover, and the open admission nature of these systems allows malicious behaviour by nodes. Performing efficient computation in dynamic networks is much more challenging than in traditional static networks. First, one has to deal with "failures" (nodes getting inserted or deleted or communication links changing continuously) as part of the "normal" mode of operation rather than as exceptions. Second, time and communication constraints are much more severe, and hence it will be too expensive or even infeasible to run a static algorithm from scratch every time that the topology changes. Dynamic networks especially need efficient distributed algorithms, since unlike static networks, pre-processing (via centralized algorithms) is not usually possible. Furthermore, nodes typically have only local knowledge (which is also changing continuously with changing neighbours and links) which necessitates *localized distributed* algorithms. Hence a new theory is needed to understand and perform robust, efficient, and secure distributed computation in such systems.

Much of the well-established theory of distributed algorithms (see e.g., [83, 97, 9, 60]) — developed over the last three decades — has focused on *static* networks (where nodes do not crash and edges maintain operational status forever). Dynamic networks pose non-trivial challenges in solving even basic and fundamental distributed computing problems, such as agreement, leader election, broadcasting, search, and routing. These problems are basic building blocks in distributed computing and are widely used (see e.g., [83, 97, 9, 41]). Unfortunately, distributed algorithmic techniques developed for static networks are not readily applicable to dynamic networks [15]. Therefore, we need rigorous models and distributed algorithmic techniques for dynamic networks.

P2P Networks. An important motivating example is P2P computing which has emerged as one of the key networking technologies with many application systems, e.g., Skype, BitTorrent, CrashPlan, Symform, Cloudmark, Bitcoin etc. Systems such as CrashPlan [42] and Symform [107] are relatively recent P2P-based storage services that allow data to be stored and retrieved among peers [96]. Such peer-based data sharing avoids costly centralized storage and retrieval and is inherently scalable. However, these systems are not

[*]Dept. of Computer Science and Engineering, IIT Madras, Chennai, India 600036. Email: `augustine@iitm.ac.in`. Supported by IIT Madras New Faculty Seed Grant, IIT Madras Exploratory Research Project, and Indo-German Max Planck Center for Computer Science (IMPECS).

[†]Department of Computer Science, University of Houston, Houston, TX 77204, USA. E-mail: `gopalpandurangan@gmail.com`. Supported, in part, by NSF grants CCF-1527867 and CCF-1540512.

[‡]Department of Computer Science, Royal Holloway, University of London, UK. E-mail: `peter.robinson@rhul.ac.uk`.

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. Cloudmark [38] is a large spam detection system used by millions of people that operates by maintaining a hybrid P2P network; it uses a central authority to regulate and charge users for participation in the network. 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. Indeed, P2P networks are highly dynamic networks characterized by high degree of node *churn* — i.e., nodes continuously join and leave the network. Connections (edges) may be added or deleted at any time and thus the topology changes very dynamically. In fact, measurement studies of real-world P2P networks [53, 63, 104, 106] show that the *churn rate* (i.e. the number of nodes that join/leave the network per *unit time or round*) is quite 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 [96, 50, 49, 33, 65], collaborative filtering [30], spam detection [38], data mining [44], worm detection and suppression [85, 109], privacy protection of archived data [59], and recently, for cloud computing services as well [107, 24]. 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, which can be as much as linear (in the stable network size). This is a major obstacle in implementation and wide-spread use of P2P systems.

Distributed Computation in Dynamic Networks. Performing efficient distributed computation in highly dynamic networks where both nodes and edges can change continuously by a large amount is a major challenge. Indeed, when the churn rate is linear in the network size, in *constant* 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., [3, 48, 58, 4, 18, 23]). Recently, dynamic networks have been considered in the context of distributed computation, mostly assuming the model of [91, 75, 76]. In this setting, algorithms and complexity bounds for problems like distributed consensus [77], random walks [43, 16], token dissemination [51, 43], aggregation [40], and counting [2] have been developed. (Section 5 has pointers to several recent works in dynamic networks.) Most of these works assume that the set of participating nodes is *fixed*. In contrast, much less is known about dynamic networks with *churn*, i.e., where the set of participants can change over time. Thus it is important to develop a rich theory of dynamic networks with high churn rate which is the focus of this article.

Byzantine Protocols. Besides dynamism, another major challenge is dealing with malicious (also called *Byzantine*) nodes in the network which can try to foil the distributed protocol executed by honest (good) nodes. Byzantine protocols are at the heart of secure and robust protocols that can tolerate the presence of malicious nodes in a distributed system, such as a P2P network, which allows a large number of peers to enter the network with little or no admission control. Such malicious peers acting alone or in collaboration can cause disruption of service in P2P systems [1, 34, 81]. Designing scalable Byzantine protocols that work in highly dynamic networks is significantly more challenging compared to static networks. Indeed, until recently, almost all the work known in the literature (see e.g., [52, 69, 71, 73, 108, 74]) has addressed the Byzantine agreement and leader election problem (defined in Section 3) only in static (bounded-degree) networks, and these approaches *do not work* for dynamic networks with changing topology. Such approaches fail in dynamic networks where both nodes *and* edges can change by a large amount in *every* round. For example, the work of [108] showed how one can achieve almost-everywhere agreement[1] under up to a *linear* number — up to ϵn, for a sufficiently small $\epsilon > 0$ — of Byzantine faults in a bounded-degree expander network (n is the network size). The algorithm requires $O(\log n)$ rounds and polynomial (in n) number

[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 [52].

of messages; however, the local computation required by each processor is exponential. Furthermore, the algorithm requires knowledge of the global topology, since at the start, nodes need to have this information "hardcoded". The work of [74] is important in the context of P2P networks, as it was the first to study scalable (polylogarithmic communication and number of rounds) algorithms for Byzantine agreement and leader election. However, as pointed out by the authors, their algorithm works only for static networks; here also the nodes require hardcoded information on the network topology to begin with, and thus the algorithm does not work when the topology changes. In fact, this work ([74]) raised the open question of whether one can design agreement protocols that can work in highly dynamic networks with a large churn rate.

Consider the real-world example of Bitcoin — a fully-decentralized P2P- based digital money with no central authority or middlemen [28]. A crucial aspect of Bitcoin is a computational mechanism that allows fault-tolerant Byzantine agreement among all honest nodes to agree on all the transactions, despite the presence of malicious nodes. This is a costly operation (called *mining*) that only nodes with very high computational power can perform; furthermore there is non-trivial probability of failure leading to corruption of the system. Hence, distributed computation in P2P systems must be robust to dynamism and failures, and also attack- resistant, besides being efficient and scalable.

Focus of this article. Motivated by the above considerations, we focus on studying the following fundamental distributed computing problems in dynamic networks (these are formally defined in Section 3):

1. *Agreement*: Nodes have to agree on a common value.
2. *Search and Storage*: Find data and resources in the network; store data reliably and securely.
3. *Byzantine Agreement*: Solve agreement problem under the presence of malicious nodes.
4. *Byzantine Leader Election*: Solve leader election problem under the presence of malicious nodes.
5. *Expander Maintenance*: Maintain an expander under adversarial dynamics.

In this article, we will give an overview of our recent results [15, 12, 11, 13, 14] that make progress towards developing a distributed algorithmic theory of dynamic networks. These results are steps toward designing provably efficient and robust algorithms for solving fundamental distributed computing problems in highly dynamic networks (with churn), especially P2P networks. First, we will present a rigorous theoretical framework for studying dynamic networks by presenting a model for dynamic networks with churn. Then we will present efficient techniques and algorithms for solving agreement, Byzantine agreement, Byzantine leader election, search/ storage, and expander maintenance. The algorithms presented are the first-known, fully-distributed algorithms for the above problems that work under highly dynamic settings, i.e., *high adversarial churn rates*. Furthermore, they are *localized* (i.e., do not require any global topological knowledge) and *scalable*. The algorithms and techniques can serve as building blocks for implementing other non-trivial distributed computing tasks in highly dynamic P2P networks. The techniques include efficient information spreading, support estimation (estimating aggregate functions), and random-walk based methods. The article discusses how these can be adapted to work even under highly dynamic networks with provable guarantees. The techniques presented are lightweight, local, and scalable which are essential for performing efficient and robust distributed computation in highly dynamic networks. Since the dynamic network model is adversarial, the results are also applicable to a wide range of scenarios, in particular, to less stringent (and possibly more realistic) settings. The results and techniques are also applicable to dynamic network models where only edges changes (and nodes remain fixed) as well as to static networks (where nodes can fail or be Byzantine).

A main goal of this article is to present the results in a unified way and highlight the techniques involved—this can be helpful in tackling other problems in dynamic networks with churn. Moreover, the techniques (and results) also apply seamlessly to dynamic networks (without churn) as well as to static networks, yielding new insights to well-studied problems. We also compare and contrast the above results with several related results in dynamic networks (Section 5). We summarize and discuss key open problems in Section 6.

2 A Dynamic Network Model with Churn (DNC model)

The following model for dynamic networks that incorporates churn was first proposed in [15] and has since been used in subsequent works with suitable modifications [12, 11, 10, 13, 14]. Henceforth, we will call the model discussed below as the *DNC (Dynamic Network with Churn)* model.

Apart from nodes joining and leaving the network or failing over time or behaving maliciously, the evolution of the network might also be triggered by coordinated attacks (e.g. denial-of-service attacks or wireless jamming) with the goal of corrupting the communication in the network. In a large-scale network, it is unrealistic to assume that all nodes faithfully execute the given protocol. Thus we must design scalable algorithms that can handle *adversarial* dynamic changes, are robust against failures and can tolerate malicious (i.e. *Byzantine* nodes [95]) that might collude and send corrupted messages. In particular, assume that the dynamism is controlled by an adversary that comes in two flavors: (1) *oblivious* —- has complete knowledge and control of what nodes fail or join and leave and at what time and has unlimited computational power — but is oblivious to the random choices made by the algorithm and (2) *adaptive* — in addition to the above it has knowledge of past random choices (but not current and future ones). Byzantine nodes, on the other hand, have knowledge about the entire state of network at every round (including random choices made by all the nodes) and can behave arbitrarily. However, they can communicate only through the (current) edges of the dynamic network and cannot send messages to nodes which are not neighbors at that point in time. The two key parameters of the DNC model are:

- *churn rate*: the number of nodes that can join or leave the network.
- number of Byzantine nodes (at any given time).

The DNC model is represented by a graph with a dynamically changing topology (both *nodes and edges* change from round to round) whose nodes execute a distributed algorithm and whose edges represent connectivity in the network. Communication is synchronous and occurs via message passing [97]. The computation is structured into synchronous *rounds*, i.e., nodes run at the same processing speed and any message that is sent by some node u to its neighbors in some round $r \geqslant 1$ will be received by the end of round r. In a distributed setting, where transmission of messages can be orders of magnitude slower than local computation, the expensive resource is communication. In this model, local computation (happening at a node) is thus assumed to be free. (In all our algorithms, however, local computation has polynomial (in n) complexity.) The focus will be on time (number of communication rounds) and number of messages — the two traditional complexity measures of distributed algorithms [97]. We typically assume small-sized messages, e.g., each message is of size $O(\text{polylog}\, n)$ bits, unless otherwise stated. For the bounds considered in this article, we mainly focus on time bounds, assuming that $O(\text{polylog}\, n)$-sized messages can be sent over a link in one round (i.e., the CONGEST($\text{polylog}\, n$) model [97]). *A main objective in this research is to design fast algorithms (running in a small, say, $O(\text{polylog}\, n)$ rounds) that use small-sized messages that can tolerate a large amount of churn and Byzantine nodes.* It is important that algorithms be fast, since the network itself changes dynamically by a large amount in *every* round.

The dynamic network is formally represented by a graph process $\mathcal{G} = (G^0, G^1, \ldots)$ where $G^r = (V^r, E^r)$. Each node has an ID and at any round r, the ID's of nodes in V^r are distinct. Edges can change arbitrarily in each round, but the underlying graphs are assumed to be connected and typically satisfy other properties. For example, in the works of [15, 12, 11], each G_i is assumed to be a d-regular *expander* as the high expansion allows one to tolerate a high amount of churn. In fact, expander graphs have been used extensively as candidates to solve the agreement and related problems in bounded degree graphs even in static settings (e.g., see [52, 69, 71, 73, 108]). It turns out that similar expansion properties are beneficial in the more challenging setting of dynamic networks. Expander graphs are reasonable models in the context of P2P networks.[2] Note that this model is quite general in the sense that, except for the expander property, no other special properties are assumed. Indeed, 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., [80, 92, 93]). (Section 3.4 addresses how the assumption of having an underlying expander can be maintained over time despite large amounts of ongoing churn; this itself is an important issue.) Many works on dynamic network models make similar assumptions on preservation of topological properties — such as connectivity, expansion etc. — at every step under dynamic *edge* insertions/deletions — cf. Section 5. The issue of how such properties are preserved are abstracted away from the model, which allows one to focus on the dynamism. Indeed, this abstraction has been a feature of most dynamic models e.g., see the survey of [31].

Let n be the stable network size. Nodes might be subjected to churn, which means that in each round, up to $C(n)$ nodes can be replaced by new nodes; $C(n)$ is called the *churn rate*. For all rounds $r \geqslant 0$, $|V^r \setminus V^{r+1}| = |V^{r+1} \setminus V^r| \leqslant C(n)$. Churn can also capture node failures; these are nodes that need not be churned out, but simply fail and are being replaced by new nodes. While many of the results that we discuss hold when the network size undergoes some moderate (i.e., $\pm o(n)$ nodes) changes, we assume for simplicity that the total number of nodes in the network remains n throughout the entire execution. Furthermore, edges can change *arbitrarily in each round* (as determined by an adversary); this captures in a very general fashion link availability over time.

The DNC model generalizes the more specialized *churn-free dynamic model* where the communication topology (i.e., the set of edges) varies over time but the set of nodes remains *fixed*. This model has received a lot of recent attention, see e.g., [91, 75, 76]. Thus the results and techniques discussed for the DNC model also apply to the churn-free model when assuming expansion properties on the graph sequence. In particular, it is important to note that they also apply to static expander networks where the set of nodes and edges are static, but some subset of the nodes may fail or be Byzantine.

In addition to churn, the DNC model allows up to $B(n)$ nodes to be *Byzantine* that can deviate arbitrarily from the given protocol. (Typically, we will assume that $B(n) \leqslant O(n^{\epsilon})$, where ϵ is a small positive constant.) We say that a node u is *honest* if u is not a Byzantine node and use V_{corr} to denote the set of honest nodes in the network. We can think of Byzantine nodes as being controlled by an adaptive adversary, in the sense that they know the entire states of all nodes at the beginning of every round and thus can take the current state of the computation into account when determining their next action. This setting is commonly referred to as the *full information model* (e.g., [26]). We consider the usual assumption that Byzantine nodes cannot fake their identity, i.e., if a Byzantine node w sends a message to nodes u and v, then both u and v can uniquely identify w as the sender of their received message. Note that this does not stop Byzantine nodes from forwarding fake messages on behalf of other nodes as we do not assume any authentication service. The churn along with Byzantine nodes captures node failures as well in a natural way and can model different kinds of networks such as P2P as well as ad hoc and sensor networks.

We assume a global synchronous clock such that if a node u enters the network at some later round r, then u knows the number of rounds that have passed since the start of the computation. However, any other information about the network at large is only learned through the messages that u receives and nodes do not have any a priori knowledge about who their neighbors will be in future rounds.

We now describe the assumptions and limitations of different adversaries in the DNC model. In every round, the *churn adversary* determines which nodes are being replaced by new nodes, whereas the rewiring of the connection links between nodes is under the control of the *topology adversary*. Since we are dealing with a dynamic network where edges *and* nodes can change from round to round, one can separate these kinds of changes given by the *churn adversary* and the *topology adversary*. The following describes the sequence of events that occur in each round $r \geqslant 1$. Firstly, the churn adversary chooses up to $C(n)$ nodes to be replaced by honest nodes, yielding the set V^r. We also consider the *oblivious churn adversary*, which is

[2]However, for other applications, such as ad hoc mobile networks, the underlying graph is better modelled as a geometric graph, e.g., a random geometric graph or a grid graph.

oblivious to the random choices of the nodes; note that this is equivalent to an adversary that fixes the entire sequence of graphs in advance of the algorithm. Next, the topology adversary determines the new edge set E^r. We distinguish between adaptive and oblivious topology and churn adversaries. The former can observe the entire state of the network in every round r (including all the random choices made until round $r-1$), and then modifies the graph for the next round. For example, an adaptive topology adversary can rewire the edges for round r after inspecting the states of all nodes whereas the oblivious topology adversary must specify the entire sequence of topological changes in advance before the algorithm is executed. It is important to note that Byzantine nodes are always adaptive and unaffected by whether the topology and churn adversaries are adaptive or oblivious.

After the adversaries have made their moves, the algorithm operates on the graph $G^r = (V^r, E^r)$ in round r. Each node u becomes aware of its current neighbors in G^r, can perform local computation and is able to reliably exchange messages with its neighbors according to the edges in E^r.

3 Fundamental Problems and Algorithms: An Overview

As mentioned in Section 1, the focus is on design and analysis of distributed algorithms for fundamental distributed computing problems in dynamic networks (DNC model). These are described in detail in the following sections.

3.1 Almost Everywhere (AE) Agreement and Byzantine Agreement

We consider the following version of the Agreement or the *Consensus* problem, that has been the subject of intense research for over three decades (see e.g., [41, 83, 9, 15] and the references therein). We note that AE Agreement is non-trivial in the DNC model due to churn (even without the presence of Byzantine nodes). In addition, we study AE BYZANTINE AGREEMENT which is AE Agreement but under the presence of Byzantine nodes as well as churn.

Let each node $v \in V^0$ have an associated input value from some value domain of size m; subsequent new nodes come with value $\perp$. Let $\mathcal{V}$ be the set of all input values associated with nodes in V^0 at the start of round 1. Every node u is equipped with a special decision variable $decision_u$ (initialized to $\perp$) that can be written at most once. We say that a node u *decides on* VAL when u assigns VAL to its $decision_u$. Note that this decision is irrevocable, i.e., every node can decide at most once in a run of an algorithm. As long as $decision_u = \perp$, we say that u is *undecided*. AE BYZANTINE AGREEMENT requires that a large fraction of the nodes come to a stable agreement on one of the values in $\mathcal{V}$. More precisely, *an algorithm solves* AE BYZANTINE AGREEMENT *in R rounds*, if it exhibits the following characteristics in every run:

Almost Everywhere Agreement: We say *the network has reached almost everywhere agreement by round R on the decision value* VAL* if at least $n - O(C(n) + B(n))$ honest nodes in V^R have decided on the same value VAL$^* \in \mathcal{V}$.

Validity: If all honest nodes in V^0 start with VAL $\in \mathcal{V}$, then VAL is the only admissible decision value for all but $O(B(n) + C(n))$ nodes.

Stability: Let R be the earliest round where nodes have reached almost everywhere agreement on value VAL*. We say that an algorithm *reaches stability by round R* if, at every round $r \geqslant R$, all but $O(B(n)+C(n))$ nodes in V^r have decided on VAL*.

3.1.1 Algorithms

The work of [15] addresses the AE agreement problem in dynamic networks (DNC model with no byzantine nodes). Its main contribution is an *efficient and scalable* — i.e., each node processes and sends only *polylogarithmic* (in n, the stable network size) messages per round, and local computation per node is also polylogarithmic — randomized distributed algorithm that guarantees AE agreement with high probability[3] even under high *adversarial* churn rate — up to *linear* in n per round — in a *polylogarithmic* ($O(\log^3 n)$) number of rounds. Thus this result shows that agreement is possible even under linear (in the size of n) amount of churn. The above result works under an *oblivious* churn adversary (where the sequence of graphs are determined in advance). Under an *adaptive* adversary it is shown that the amount of churn that can be tolerated is less, up to $\varepsilon\sqrt{n}$. (Moreover, this algorithm requires polynomial sized messages per round.) This is essentially the best possible amount of churn that can be tolerated under an adaptive adversary, if one requires fast algorithms (i.e., running in polylogarithmic number of rounds) [12].

The work of [12] further extends the work of [15] and gives an efficient distributed algorithm for AE Byzantine agreement that works despite the presence of *Byzantine nodes* and high adversarial churn. (It is important to note that the algorithms and techniques of [15] will fail to work if there is even one Byzantine node.) This algorithm can 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 above works presented general techniques for distributed computation in dynamic networks with churn. The first technique was information spreading via "flooding". In an expander-based dynamic network, even under linear churn rate, it is possible to spread information by flooding if sufficiently many (a β-fraction of the order of the churn, for some suitable $\beta > 0$) nodes initiate the information spreading [15]. In other words, even an *adaptive* churn adversary cannot "suppress" more than a small fraction of the values. The second technique presented was "support estimation" which is a *randomized* technique that can be used to estimate the aggregate count (or sum) of values of all or a subset of nodes in the network. Support estimation is done in conjunction with flooding and uses properties of the exponential distribution (similar to [39, 87]). Support estimation allows one to estimate the aggregate value quite precisely with high probability even under *linear* churn in a *scalable* way. But this works only for an oblivious adversary; to get similar results for the adaptive case, we need to increase the amount of bits that can be processed and sent by a node in every round to a polynomial in n.

The Byzantine agreement algorithm, crucially uses *random walk- based* techniques [12]. Random walks are very simple, local, and robust and ideally suited for highly dynamic networks [101]. Random walks are very local and lightweight and requires little index or state maintenance that makes it attractive to self-organizing networks such as Internet overlay and ad hoc wireless networks [112, 29]. Flooding and support estimation techniques are useful in solving the agreement problem under high churn but without Byzantine nodes [15], but they fail in the presence of Byzantine nodes who can send a lot of (useless) messages along with those of good nodes. On the other hand, if good nodes use random walks in their information spreading protocol, the Byzantine nodes are no longer able to flood the network very much. This proves crucial for getting a scalable protocol that uses only polylogarithmic message sizes per round and finishes in polylogarithmic rounds. The basic idea of random walks is simple: use random walks to sample tokens (approximately) uniformly at random. All nodes generate tokens (which contain the source node's ID) and send those tokens 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. While this is easy to establish in a static network, it is no longer true in a dynamic network with Byzantine nodes and adversarial churn — these can cause many random walks to be lost and also introduce bias. A key technical result called the *Dynamic Sampling Theorem* [12, 13] (see Section 4) shows that "most" random walks do mix (despite byzantine nodes and large adversarial churn) and have the usual desirable properties as in a static network. The dynamic sampling theorem allows almost uniformly sampling among the honest nodes. For solving Byzantine agreement, a majority rule is used to progress towards agreement.

It is important to note that random walk techniques work only under the *topology-oblivious* adversary;

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

on the other hand, if the adversary is topology-adaptive it can easily foil the progress of random walks. For the topology-adaptive adversary, it is shown that information spreading techniques can be used, under the additional assumption that nodes can directly contact any other node if it knows the ID of that node[4]. This "overlay" link is used as a verification mechanism to authenticate the message sent by honest nodes. This algorithm is not scalable as it requires up to polynomial in n bits to be processed and sent (per round) by each node.

3.1.2 A Lower Bound

The bound of $\tilde{\Omega}(\sqrt{n})$ on the amount of adaptive churn is essentially the best possible, if one wants fast (i.e., polylogarithmic run time) algorithms [12]. The proof of the lower bound result relies neither on the nature of the connectivity nor on the presence of Byzantine nodes. The lower bound proof builds on well-established ideas used in several consensus lower bound proofs [9]. It uses coin flipping games which were first studied by Ben- Or and Linial [27]. Subsequently, these coin flipping games have been used several times to prove lower bounds on running time of distributed consensus algorithms; see Aspnes [6] along with the references therein. The proof structure is similar to the work by Ben-Or and Bar-Joseph [26], which in turn builds on [57, 47, 6, 27]. We refer the reader to [12] for more details.

3.2 Leader Election

Leader Election and Byzantine Leader Election (BLE) are central problems in distributed computing and have been extensively studied see e.g., [79, 13, 74]. We say that an *algorithm A solves Byzantine Leader Election in T rounds* if, in any run of A, there is exactly 1 node u_ℓ such that

(a) all honest nodes terminate in T rounds whp,
(b) all except $o(n)$ honest nodes accept u_ℓ as the leader, and
(c) node u_ℓ is honest with probability $\geqslant 1 - o(1)$.

3.2.1 A Byzantine Leader Election Algorithm

In the churn-only DNC model (where all nodes are honest), Leader Election problem can be solved by solving Agreement (nodes will agree on the ID of some particular node); thus the algorithm of [15] will work. However, in the Byzantine setting, an algorithm for AE BYZANTINE AGREEMENT does not directly solve the BLE problem, since the value that (most of) the honest nodes agree may be a value that was generated by a Byzantine node; using the agreement algorithm in a straightforward way does not give any guarantee that an honest node will be elected as leader. Hence, a more involved approach is needed for Byzantine leader election.

The work of [13] presents a scalable ($O(\log^3 n)$ round) BLE protocol in the *topology-oblivious* model that can tolerate only up to $O(n^{\frac{1}{2}-\epsilon})$ Byzantine nodes (for a small constant $\epsilon > 0$) and up to $O(\sqrt{n}/\operatorname{polylog}(n))$ churn rate. We outline the main ingredients of the protocol. While Byzantine agreement itself does not directly help, it can be used to generate an almost-everywhere public coin, i.e., an almost fair public coin that is known to most honest nodes. The protocol requires nodes to independently generate "lottery tickets" which are bit strings of certain length. Essentially, a node that has the winning lottery ticket becomes part of the small set of finalists from which a leader will be chosen eventually. Note that the lottery tickets are generated first. However, there is a problem in naively implementing this approach. The Byzantine nodes who know the current network state *including* random choices of other nodes can lie about their lottery ticket number, thus claiming to be the winner. To overcome this, we implement a *verification* mechanism that allows the honest nodes to check whether the Byzantine nodes are lying. This mechanism is as follows. Once

[4]This is a typical assumption in P2P and overlay networks.

a node generates its lottery ticket it "stores" it in about $\sqrt{n}$ (randomly chosen) nodes (exceeding the number of Byzantine nodes by an n^ϵ factor). To verify whether a node is indeed the owner of the lottery ticket that it claims, honest nodes will check with these $\sqrt{n}$ nodes. This prevents a Byzantine node from falsely claiming a lottery ticket that it did not generate in the first place. Such a storage and verification mechanism can be implemented efficiently despite the presence of Byzantine nodes in a dynamic network. The last ingredient of the protocol is an efficient and fault-tolerant mechanism to disseminate the identity of the leader to almost all the honest nodes. This is done using random walks which is (again, as in the case of Byzantine agreement) a main technical tool used throughout the algorithm.

A drawback of the above protocol is that it can tolerate only up to $O(n^{\frac{1}{2}-\epsilon})$ Byzantine nodes (for a small constant $\epsilon > 0$) and up to $O(\sqrt{n}/\operatorname{polylog}(n))$ churn rate. This is essentially the best that our protocol can handle for two reasons. Our storage and verification mechanism needs to store each lottery ticket in about $\sqrt{n}$ nodes, and to be scalable in terms of the number of messages generated, it can handle only about $\sqrt{n}$ nodes (each such node generates a ticket, thus overall there will be about $n \operatorname{polylog}(n)$ messages — anything significantly more than this will cause much more congestion). The second reason is that for solving Byzantine agreement, we use a majority rule to progress towards agreement [46, 12]. This majority rule algorithm works as long as the number of Byzantine nodes is bounded by $O(\sqrt{n})$.

3.3 Storage and Search

The goal is to build a robust distributed solution for the storage and retrieval of data items. Nodes can produce data items. Each data item is uniquely identified by an ID (such as its hash value). When a node produces a data item, the network must be able to place and maintain copies of the data item in several nodes of the network. To ensure scalability, we want to upper bound the number of copies of each data item, but more importantly, we must also replicate the data sufficiently to ensure that, with high probability, the churn does not destroy all copies of a data item. When a node u requires a data item (whose ID, we assume, is known to the node), it must be able to access the data item within a bounded amount of time. To keep things simple, we only require that u knows the ID of *a* node (currently in the network) that has the data item u needs.

While many P2P systems/protocols have been proposed for efficient search and storage of data (see e.g., [82, 11] and the references therein), a major drawback of almost all these is the lack of algorithms that work with provable guarantees under a large amount of churn per round. In such a highly dynamic setting, it is non-trivial to even just store data in a persistent manner; the churn can simply remove a large fraction of nodes in just one time step. On the other hand, it is costly to replicate too many copies of a data item to guarantee persistence. Thus the challenge is to use as little storage as possible and maintain the data for a long time, while at the same time designing efficient search algorithms that find the data quickly, despite the high churn rate.

The work of [11] presented efficient randomized distributed algorithms for searching, storing, and maintaining data in dynamic P2P networks. These algorithms succeed with high probability (i.e., with probability $1 - 1/n^{\Omega(1)}$, where n is the stable network size)) even under high adversarial churn in a polylogarithmic number of rounds. In particular, the following algorithms are presented:

1. A storage and maintenance algorithm that guarantees, 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*, assuming that the churn is controlled by an oblivious adversary.

2. A randomized distributed search algorithm that with high probability guarantees that searches from as many as $n - o(n)$ nodes succeed in $O(\log n)$-rounds under up to $O(n/\operatorname{polylog}(n))$ churn *per round*. The search algorithm together with the storage and maintenance algorithm guarantees that as many as $n - o(n)$ nodes can efficiently store, maintain, and search even under $O(n/\operatorname{polylog}(n)))$ churn

per round. The above algorithms require only polylogarithmic in n bits to be processed and sent (per round) by each node.

The above search and storage algorithms crucially use random walks as well. In particular, they also use the Dynamic Sampling Theorem (referred to as the *Soup Theorem* in [11]) which is discussed in Section 4.3. The Dynamic Sampling Theorem allows robust and accurate sampling of nodes despite high adversarial churn—upto $O(n/\operatorname{polylog}(n))$ per round. It is important to note that the random walk-based technique can handle churn only up to $n/\operatorname{polylog}(n)$. 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 our random walk based method. Note that the above search and storage algorithms that can handle up to $O(n/\operatorname{polylog}(n))$ churn do not assume any presence of Byzantine nodes. However, they can be shown to tolerate up to $O(\sqrt{n}/\operatorname{polylog}(n))$ Byzantine nodes and up to a $O(\sqrt{n}/\operatorname{polylog}(n))$ (topology-oblivious) churn rate (this follows from [13]).

Another useful technique in search and storage algorithms is construction and maintenance of (small-sized) *committees*. A committee is a clique of small ($\Theta(\log n)$) size composed of essentially "random" nodes (these random nodes are sampled using the Dynamic Sampling Theorem). A committee can be efficiently constructed, and more importantly, *maintained* under large churn. A committee can be used to "delegate" a storage or a search operation; its small size guarantees scalability, while its persistence guarantees that the operation will complete successfully despite churn.

3.4 Expander Maintenance

A crucial ingredient that underlies all the results discussed till now 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).

The assumption that the underlying topology is an expander in every round 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 [93, 80, 92] and references therein). However, none of these works guarantee the maintenance of an expander under the more challenging situation of high continuous adversarial churn.

The work of [14] presents 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 — up to $O(n/\operatorname{polylog}(n))$, where n is the stable network size. The churn is controlled by an oblivious churn adversary. 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 [5]) 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).

The expander maintenance protocol 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). The protocol guarantees that, despite a large adversarial churn, with high probability, there is a *large expander subgraph ($n - o(n)$ size)* present in the network in every round. The degree of each node is always bounded

[5]The 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., $O(n/\text{polylog } n)$).

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[6]; after this, the adversary is free to exercise its power to churn in/out nodes up to the churn limit.

This is the first-known, fully-distributed protocol that guarantees expansion maintenance under highly dynamic and adversarial settings. The protocol is local (does not require any global topological knowledge), simple, lightweight, and easy to implement. The protocol serves as a building block for enabling distributed 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 [15], and search and storage [11] enables a fully-distributed distributed and autonomous protocol for these problems.

The main intuition behind the protocol is that it tries to maintain a graph that resembles a random (bounded-degree) graph in every round whp. The protocol works by ensuring two interdependent invariants (whp). The first invariant is that the nodes can sample from the set of ID's in the network almost uniformly at random. This is implemented via *random walks*. For the sampling technique to work efficiently, a second invariant is needed, namely, that the network maintains good expansion. To achieves this, the protocol connects each node to a constant number of other nodes chosen (almost) uniformly at random, leading to a "random- enough" graph that resembles an expander, and thus bringing back to the 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 to be overcome in order to get the protocol to work.

We give a high-level overview of the main ideas of the protocol and some intuition for understanding its design. 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 Dynamic Sampling Theorem 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, the protocol tries to maintain a graph that resembles a random 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. 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

[6]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.

number of red edges there is a good (a constant) probability of establishing a connection in a round.

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; such sharing of tokens is problematic if continued for a long time (since tokens are 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. It is shown that nodes cannot remain cut-off for long and will reconnect to the "core" network in at most $O(\log n)$ rounds (whp).

4 Algorithmic Techniques for Dynamic Networks

We now describe a toolkit of algorithmic techniques that we can use as building blocks to solve fundamental problems in dynamic networks.

4.1 Information Spreading against a Topology-Adaptive Adversary

Recall that the adversary has the power to strategically isolate any given set of nodes S by simply subjecting all neighbors of S in $V^r \setminus S$ to churn. Nevertheless, we can show that "almost all" nodes are able to disseminate information to almost all other nodes in the DNC model. (Throughout this section, "almost all" refers to a quantity of size at least $n - O(C(n))$ nodes.) Intuitively speaking, the vertex expansion α of the underlying graph sequence effectively limits the adversary's impact as it is impossible to effectively "surround" a set that exceeds $C(n)$ by a sufficiently large fraction, since this set must have at least $\alpha C(n)$ neighbors in other parts of the network, assuming $C(n) \leqslant n/2$.

We now show how to implement an *almost everywhere broadcast* primitive in the DNC model, which intuitively means that almost all nodes are able to broadcast a message that is relayed by other nodes and reaches almost all nodes in the network within a time depending on the churn rate. To aid in the forwarding of a broadcast message, we assume that any node can generate a flooding message with an attached terminating condition. Then, the initiating node broadcasts the message to all of its current neighbors including itself. When an honest node receives a flooding message, it continues to broadcast that message in subsequent rounds as long as the terminating condition of the message is satisfied.

We first introduce the notion of influence which extends similar definitions for dynamic networks without churn (cf. [75]). The *dynamic distance from node u to node v in round r*, denoted by $\mathsf{DD}_r(u \to v)$, is the number of rounds required for at least one message originating from u in r to reach v. Note that messages can fall victim to churn if they reach a node that is churned out in the very same round. Suppose that node u joins the network in round r_u and, from round $\max(r_u, r)$ onward, u initiates a message m for flooding whose terminating condition is: $\langle$HAS REACHED $v\rangle$. If u is churned out before r, then $\mathsf{DD}_r(u \to v)$ is undefined. We define $\mathsf{DD}_r(u \to v) = \Delta$ if the first of those flooded messages reaches v in round $r + \Delta$; formally speaking this is the length of the shortest causal chain that reaches v starting in round r at u. Note that this definition allows $\mathsf{DD}_r(u \to v)$ to be infinite under two scenarios. Firstly, node v may be churned out before any copy of m reaches v. Secondly, at each round, v can be shielded by churn nodes or Byzantine nodes that absorb or corrupt the flooded messages.

The *influence set* of a node u after R rounds starting at round r is defined as $\mathsf{Influence}_r(u, R) = \{v \in V^{r+R} : \mathsf{DD}_r(u \to v) \leqslant R\}$, where V^{r+R} denotes the set of nodes in round $r + R$. Note that we require $\mathsf{Influence}_r(u, R)$ to be a subset of V^{r+R}. Intuitively, we want the influence set of u (in this dynamic setting) to capture the nodes *currently* in the network that were influenced by u. Note however that the influence set of a node u is meaningful even *after* u is churned out. When considering only a single node u, the adversary can easily prevent the influence set of this node from ever reaching any significant size by simply subjecting the neighbors of u to churn in every round. This motivates us to define the influence set of any set of nodes $U \subseteq V^r$ as

$$\mathsf{Influence}_r(U, R) \triangleq \cup_{u \in U} \mathsf{Influence}_r(u, R).$$

We say that a node $u \in V^r$ is *suppressed for* R rounds if $|\mathsf{Influence}_r(u, R)| < n - \beta C(n)$ where β is a sufficiently large constant depending on the expansion of $\mathcal{G}$; note that we assume $\beta C(n)$ to be an integer to keep the notation simple. Otherwise we say that u is *unsuppressed.*

Considering a set A_0 of $\beta C(n)$ nodes, we can leverage the graph expansion to show that the nodes in A_0 will jointly influence a large set B_0 of $\geqslant n - \beta C(n)$ nodes in $T = O(\log(\frac{n}{C(n)}))$ rounds. Note that, on its own, such a result is insufficient for achieving almost everywhere broadcast, as the joint influence of the nodes in this set does not allow us to say anything about how many nodes in B_0 are influenced by a particular node in A_0 in round $r + T$. However, since each node in B_0 is influenced by at least 1 node in A_0, a combinatorial counting argument shows that there is a subset $A_1 \subset A_0$ of size $\leqslant \frac{|A_0|}{2}$, the nodes of which jointly influence a set $B_1 \subseteq B_0$ of $\geqslant \frac{|B_0|}{2}$ nodes in B_0. As $|B_1| \geqslant \frac{n - \beta C(n)}{2} \geqslant \beta C(n)$, we can argue the same way about B_1 starting at round $r + T$ over the next T rounds as we did for set A_0 above. In other words, the nodes in B_1 will jointly influence a set of $\geqslant n - \beta C(n)$ nodes in round $r + 2T$. Repeating this argument $\Theta(\log n)$ times, allows us to obtain a sequence of sets of exponentially shrinking size until we are left with a singleton set consisting of a node $u^* \in A_0$ that influences $\geqslant n - \beta C(n)$ nodes in round $r + T \log n$. Thus we have identified an unsuppressed node within our set A_0, which is any arbitrary set of $\beta C(n)$ nodes. This implies that at most $\beta C(n) - 1$ nodes can be suppressed at any time, as any larger size set of nodes must contain at least one unsuppressed node.

Theorem 1 (Almost Everywhere Broadcast)**.** *Consider the DNC model with churn rate $C(n)$. Then, there is a set S of $\geqslant n - O(C(n))$ nodes such that, if a node in S broadcasts a message in round r, this message is received by all except $O(C(n))$ nodes in round $r + O(\log(\frac{n}{C(n)}) \log n)$.*

4.2 Support Estimation Under an Oblivious Churn Adversary

In several contexts, we need to count the number of nodes in the network that either (i) have a certain property or (ii) propose a certain value (such as in agreement protocols). In a dynamic setting, it will be impossible to count the nodes precisely. Even if the nodes with the desired property are stable, they may be suppressed due to churn. So we quite naturally resort to approximate counting, which is sufficiently useful for most purposes.

More formally, we have a DNC with churn that is up to linear in n in which an oblivious adversary has selected some $\mathcal{R}$ nodes in V^0 and colored them red. $\mathcal{R}$ is called the *support* of red nodes. We want most nodes in the network to estimate $\mathcal{R}$ under an oblivious adversary.

Our algorithm (Algorithm 4.1) uses random numbers drawn from the exponential distribution, whose probability density function recall is parameterized by λ and given by $f(x) = \lambda e^{-\lambda x}$ for all $x \geqslant 0$. The expected value of a random number drawn from the exponential distribution of parameter λ is $1/\lambda$. Our algorithm hinges primarily on an important property that relates the minimum of several exponentially distributed random variables with their parameters.

Property 1 (see [61] or [86])**.** *Consider $K \geqslant 1$ independent random variables $Y_1, Y_2, \ldots, Y_K$, each following the exponential distribution of rate λ_i. The minimum among all Y_i's, for $1 \leqslant i \leqslant K$, is an exponentially distributed random variable with parameter $\sum_{i=1}^{K} \lambda_i$.*

The idea behind our algorithm (see Algorithm 4.1 for pseudocode) exploits Property 1 in the following manner. Each of the $\mathcal{R}$ red nodes generates an exponentially distributed random number with parameter 1 (and a precision of $\Theta(\log n)$ bits) and floods its random numbers, but only propagating the minimum and eliminating all other random numbers. Since each node only needs to flood the smallest random number seen so far, the message broadcast by each node per time step is only $O(\log n)$ bits (or $O(\text{polylog } n)$ bits when $O(\text{polylog } n)$ executions are run in parallel). Out of the $\mathcal{R}$ red nodes, at least $\mathcal{R}'$ nodes of cardinality at least $\mathcal{R} - O(C(n))$ will be unsuppressed. Then the minimum $\bar{s}$ among those $\mathcal{R}'$ random numbers will also be exponentially distributed, but with parameter $\mathcal{R}'$. Thus $1/\bar{s}$ serves as an estimate of $\mathcal{R}'$, and therefore also for $\mathcal{R}$ (within $O(C(n))$). To get a more accurate estimation of $\mathcal{R}$, we can exploit sharp concentration bounds (see [87] and pp. 30, 35 of [45]) when the process is repeated (in parallel) a sufficient number of times and thereby obtaining the following claim.

```
Note: The following pseudocode is executed at every node $u$. $P \in \Theta(\log n)$ controls the precision of our estimate.

At round 0:
  1: Draw $P$ random numbers $s_1, s_2, \ldots, s_i, \ldots, s_P$, each from the exponential random distribution with rate 1.
     {Each $s_i$ is chosen with a precision of $\Theta(\log n)$ bits to ensure that the smallest possible positive value is at most $\frac{1}{n^{\Theta(1)}}$.}
  2: For each $s_i$, create a message $m_u(i)$ containing $s_i$ and a terminating condition: HAS ENCOUNTERED A MESSAGE $m_v(i)$ WITH
     A SMALLER RANDOM NUMBER.
     {Notice that a message with index $i$ will terminate only when it encounters another message with the same index, but smaller
     random number. }
  3: For each $i$, initiate flooding of message $m_u(i)$.

For the next $t = \Theta(\log n)$ rounds:
  4: Continue flooding messages respecting their termination conditions (mentioned in line number 2). I.e., among all messages of
     the form $m_*(i)$, only one message with the smallest random number will continue to flood; all others will be terminated.
     {It is easy to see that the number of bits transmitted per round through a link is at most $O(\log^2 n)$.}

At the end of the $\Theta(\log n)$ rounds (i.e., the terminating round):
  5: For each $i$, the node $u$ holds a message $m_v(i)$. Let $\bar{s}_u(i)$ be the random number contained in $m_v(i)$.
  6: $\bar{s}_u \leftarrow \frac{\sum_i \bar{s}_u(i)}{P}$.
  7: Node $u$ outputs $1/\bar{s}_u$ as its estimate of $\mathcal{R}$. {Now that the estimation is completed, all messages can be terminated.}
```

Algorithm 4.1: Algorithm to estimate the support $\mathcal{R}$ of red nodes when $\mathcal{R} \geqslant n/2$.

Theorem 2. *Consider an oblivious adversary and let γ be an arbitrary fixed constant $\geqslant 1$. Let $\bar{\mathcal{R}} = \max(\mathcal{R}, n - \mathcal{R})$. By executing Algorithm 4.1 to estimate both $\mathcal{R}$ and $n - \mathcal{R}$, at least $(1-\beta)n$ nodes in the terminating round will estimate $\bar{\mathcal{R}}$ to within $[(1-\delta)\bar{\mathcal{R}}, (1+\delta)\bar{\mathcal{R}}]$ for any $\delta > 2\beta$ with probability at least $1 - n^{-\gamma}$.*

We point out that the above technique fails when churn is controlled by an adaptive adversary (capable of observing the random numbers generated by the nodes) or when Byzantine nodes are present. An adaptive adversary can selectively churn out nodes that generate the smallest random numbers, while Byzantine nodes can fake the minimum value.

4.3 Dynamic Sampling

We now turn our attention to another prominent algorithmic tool that has played a crucial role in designing algorithms capable of handling adversarial dynamism and Byzantine faults. This random walks based tool has low overhead (of only $O(\log n)$ random walk tokens communicated over each edge per time step) and allows most nodes to sample a steady stream of $\Theta(\log n)$ nodes from the network at each time step. Apart from being lightweight, random walks work in a highly parallel and distributed manner that makes it difficult for any adversary — whether it be the adversary that controls dynamism or the adversary that controls Byzantine faults — to predict or manipulate. Although this technique has been useful in Byzantine settings [12], our focus in this survey will be limited to the DNC model without any Byzantine nodes, but a churn rate of up to $O(n/\text{polylog } n)$ nodes per time step controlled by an oblivious adversary.

```
1: for each round $r$ do
2:   for each node $v$ in parallel do
3:     Initiate $\Theta(\log n)$ random walk tokens with:
         (i) a time to live (TTL) timer set to $\tau \in \Theta(\log n)$, and
         (ii) a stamp of the source node's ID.
4:     If there are any random walk tokens with expired TTL timers, $v$ "consumes" their source node ID's
       as the required samples and discards the tokens.
5:     Move each of the remaining random walk tokens to a uniformly and independently chosen neighbor
       (after decrementing their TTL timers).
```

Algorithm 4.2: Random sampling in a dynamic network.

We will now present the random sampling algorithm (see Algorithm 4.2 for pseudocode) and sketch its analysis. In order to get a steady stream of samples, the algorithm is typically executed ad infinitum in the background so that any other protocol that requires samples can consume them as needed. At each time step, every node u in the network generates $\Omega(\log n)$ new random walk tokens that are stamped with u's ID. The random walk tokens take independent random steps (i.e., move to a neighbouring vertex with equal likelihood) each round for a period of $\Theta(\log n)$ rounds. At this point in time the tokens have mixed sufficiently and the source ID stamped into the random walk token can be "consumed" as a sample. Clearly, the samples may not all be good. For example, some of the samples may be pointing to nodes that have already churned out, thereby making such samples meaningless in the current round. However, we show that there is a large set of nodes (of cardinality $(1 - o(n))n$) called $\textsc{Core}$ such that nodes within the $\textsc{Core}$ receive good samples from other nodes within the $\textsc{Core}$. We now formally state our result and subsequently sketch its proof. (For complete proof, see [11, 14].)

Theorem 3 (Dynamic Sampling Theorem)**.** *There exists a $\tau \in \Theta(\log n)$ such that for all $r \geqslant 1$, there exists a set of nodes $\textsc{Core} \subseteq V^r \cap V^{r+\tau}$ for which the following properties hold for arbitrary choice of $s \in \textsc{Core}$ and $d \in \textsc{Core}$:*

1. *$|\textsc{Core}| \in \Omega(n/\textit{polylog}\ n)$.*
2. *The number of tokens that pass through an edge at any round is at most $O(\textit{polylog}\ n)$ with high probability.*
3. *A random walk token initiated in round r at s will terminate at d in round $r + \tau$ with probability in $\Theta(1/n)$.*
4. *Finally, as a complement to the previous property, a random walk that terminated at d in round $r + \tau$ was initiated by s with probability in $\Theta(1/n)$.*

Proof Sketch. It is of course well-known via standard spectral theoretic analysis that random walks on d-regular expander graphs will reach close to stationary distribution within $O(\log n)$ time steps. Moreover, the d-regularity will ensure that the expected number of random walk tokens is uniform across all nodes, thereby ensuring that at most polylog(n) tokens move across each edge per time step — both on expectation and with high probability. As a result, Protocol 4.2 will clearly provide a steady stream of random samples from the set of nodes when the underlying communication graph is a static d-regular expander.

There are two main issues to wrestle with when analyzing the random sampling algorithm when the underlying communication graph can be dynamic. Firstly, under churn of $O(n/\text{polylog}n)$ per time step, many random walks will be eliminated along with nodes that are churned out. Secondly, can an oblivious adversary capable of both churn *and* rewiring the graph at every time step inject significant bias? We need a more nuanced analysis to show that the dynamic sampling protocol is indeed capable of providing good samples to most nodes.

Before we grapple with churn, let us make a few observations about the random sampling when the node set is fixed. This will allow us to focus on the mixing characteristics when random walks are preserved.

Since the communication graph is a d-regular graph at each time step, the expected number of random walk tokens at each node per time step is going to stay bounded. Moreover, Das Sarma *et al.* [43] show that the standard spectral graph theoretic analysis can be applied to dynamic graphs when the node set is fixed. In particular, when each of the communication graphs is a d-regular expander with each of their second largest eigenvalues in absolute value upper bounded by $\lambda < 1$, the random walks will reach close to the uniform stationary distribution.

To extend the analysis to networks with churn, we follow a two step process. In the first step, we transform the dynamic network with churn into a virtual network without churn. We achieve this by copying the state of each node that is about to be churned out on to a node that is churned in. So in this virtual network, the random walk tokens are preserved. The results from [43] can be applied to the virtual network and from this, the mixing time characteristics of tokens in the virtual network can be inferred. The second step will be translate the random walk characteristics in the virtual network over to dynamic networks with churn. We show that a churn of $O(n/\text{polylog } n)$ nodes per time step can only have a limited impact and that most nodes will in fact receive well-mixed tokens at every time step. □

5 Other Related Work and Comparison

There has been a significant amount of work in dynamic networks in recent years. We discuss and compare those that are related to the model and algorithms outlined in this article.

5.1 Dynamic Networks

Dynamic networks have been studied extensively over the past three decades. Some of the early studies focused on dynamics that arise out of faults, i.e., when edges or nodes fail. A number of fault models, varying according to extent and nature (e.g., probabilistic vs. worst-case) and the resulting dynamic networks have been analyzed (e.g., see [9, 83]). There have been several studies on models that constrain the rate at which changes occur, or assume that the network eventually stabilizes (e.g., see [3, 48, 58]). Some of the early work on general dynamic networks include [4, 18] which introduce general building blocks for communication protocols on dynamic networks. Another notable work is the local balancing approach of [17] for solving routing and multicommodity flow problems on dynamic networks. Most of these papers develop algorithms that will work under the assumption that the network will eventually stabilize and stop changing.

Modeling general dynamic networks has gained renewed attention with the recent advent of heterogeneous networks composed out of ad hoc, and mobile devices. To address highly unpredictable network dynamics, stronger adversarial models have been studied by [16, 43, 91, 77] and others; see the recent survey of [31] and the references therein. The works of [77, 16, 43] study a model in which the communication graph can change completely from one round to another, with the only constraint being that the network is *connected at each round* ([77] and [43] also consider a stronger model where the constraint is that the network should be an expander or should have some specific expansion in each round). The model has also been applied to agreement problems in dynamic networks; various versions of coordinated consensus (where all nodes must agree) have been considered in [77]. The recent work of [37], studies the flooding time of *Markovian* evolving dynamic graphs, a special class of evolving graphs.

The model of [76] allows only edge changes from round to round while the nodes remain fixed. On the other hand, this article focuses on a dynamic network model where both nodes and edges can change by a large amount. Therefore, this framework is more general than the model of [76], as it is additionally applicable to dynamic settings with node churn. According to [75], coping with churn is one of the important open problems in the context of dynamic networks.

An important aspect of the algorithms discussed here is that they will work and terminate correctly even when the network keeps continually changing. There has been considerable prior work in dynamic P2P networks (see [92] and the references therein) but these do not assume that the network keeps continually changing over time. On the other hand, an important aspect of the protocols outlined here is that they work even when the network is continually changing in an highly dynamic and adversarial manner. Most prior algorithms (e.g., [80, 67, 94, 93, 20, 84]) 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.

Due to the mobility of nodes, mobile ad-hoc networks can also be considered as dynamic networks. The focus of [91] are the minimal requirements that are necessary to correctly perform flooding and routing in highly dynamic networks where edges can change but the set of nodes remains the same. In the context of agreement problems, electing a leader among mobile nodes that may join or leave the network at any time is the focus of [36, 35]. To make leader election solvable in this model, Chung et al. introduce the notion of D-connectedness, which ensures information propagation among all nodes that remain long enough in the network. Note that, in contrast to the model discussed here, this assumption prohibits the adversary from permanently isolating parts of the network. The recent work of [64] presents information spreading algorithms on dynamic networks (with no churn) based on network coding [5].

5.2 Fault-Tolerance

In most work on fault-tolerant agreement problems the adversary a priori commits to a fixed set of faulty nodes. In contrast, [46] considers an adversary that can corrupt the state of some (possibly changing) set of $O(\sqrt{n})$ nodes in every round. The median rule of [46] provides an elegant way to ensure that most nodes stabilize on a common output value within $O(\log n)$ rounds, assuming a complete communication graph.

Expander graphs and spectral properties have already been applied extensively to improve the network design and fault-tolerance in distributed computing (cf. [108, 52, 25]). Law and Siu [80] provide a distributed algorithm for maintaining an expander in the presence of churn with high probability by using Hamiltonian cycles. In [94] it is shown how to maintain the expansion property of a network in the self-healing model where the adversary can delete/insert a new node in every step. In the same model, [93] 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. In [8], it is shown that a SKIP graph (cf. [7]) contains a constant degree expander as a subgraph with high probability. Moreover, it requires only constant overhead for a node to identify its incident edges that are part of this expander. Later on, [67] presented a self-stabilizing algorithm that converges from any weakly connected graph to a SKIP graph in time polylogarithmic in the network size, which yields a protocol that constructs an expander with high probability. In [20] 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 [84] 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. Note that, however, the expansion in [84] is only guaranteed with high probability however, assuming that the node degree is $\Omega(\log n)$.

In the context of maintaining properties in P2P networks, Kuhn et al. consider in [78] that up to $O(\log n)$ nodes can crash or join per constant number of time steps. Despite this amount of churn, it is shown in [78] how to maintain a low peer degree and bounded network diameter in P2P systems by using the hypercube and pancake topologies. Scheideler and Schmid show in [103] 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 [22], 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. Note that, however, collisions are likely to occur once the number of attacks becomes $\Omega(\sqrt{n})$.

There has been a lot of work on P2P algorithms for maintaining desirable properties (such as connectivity, low diameter, bounded degree) under churn (see e.g., [92, 68, 78], but these don't work under large adversarial churn rates. There has also been significant work in the design of P2P systems for doing efficient search. These can be classified into two categories — (1) Distributed Hash Table(DHT)-based schemes (also called "structured" schemes) and (2) unstructured schemes; we refer to [82] for a detailed survey. However much of these systems have no provable performance guarantees under large adversarial churn. There has been works on building fault-tolerant Distributed Hash Tables (which are classified as "structured" P2P networks unlike ours which are "unstructured" e.g., see [92]) under different deletion models — adversarial deletions and stochastic deletions. The structured P2P network described by Saia *et al.* [100] guarantees that a large number of data items are available even if a large fraction of *arbitrary* peers are deleted, under the assumption that, at any time, the number of peers deleted by an adversary must be smaller than the number of peers joining. Naor and Weider [90] 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 [66] describe how to modify two popular DHTs, Pastry [99] and Tapestry [111] to tolerate random deletions. Several DHT schemes (e.g., [105, 98, 70]) 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 [88, 89]). However, these do not apply to our model of continuous churn.

5.3 Distributed Agreement and Byzantine Agreement

The distributed agreement (or consensus) problem is important in a wide range of applications, such as database management, fault-tolerant analysis of aggregate data, and coordinated control of multiple agents or peers. There is a long line of research on various versions of the problem with many important results (see e.g., [9, 83] and the references therein). The relaxation of achieving agreement "almost everywhere" was introduced by [52] in the context of fault-tolerance in networks of bounded degree where all but $O(t)$ nodes achieve agreement despite $t = O(\frac{n}{\log n})$ faults. This result was improved by [108], which showed how to guarantee almost everywhere agreement in the presence of a linear fraction of faulty nodes. Both the work of [52, 108] crucially use expander graphs to show their results.

We note that Byzantine adversaries are quite different from the adversaries considered in this article. A Byzantine adversary can have nodes behaving arbitrarily, but no new nodes are added (i.e., no churn), whereas in this case (an external) adversary controls the churn and topology of the network but *not* the behavior of the nodes. Despite this difference it is worthwhile to mention that there has been significant work in designing peer-to-peer networks that are provably robust to a large number of Byzantine faults [55, 66, 90, 102]. These focus only on robustly enabling storage and retrieval of data items. The problem of achieving almost-everywhere agreement among nodes in P2P networks (modeled as an expander graph) is considered by King et al. in [74] in the context of the leader election problem; essentially, [74] is a sparse (expander) network implementation of the full information protocol of [73]. More specifically, [74] assumes that the adversary corrupts a constant fraction $b < 1/3$ of the processes that are under its control throughout the run of the algorithm. The protocol of [74] guarantees that with constant probability an uncorrupted leader will be elected and that a $1 - O(\frac{1}{\log n})$ fraction of the uncorrupted processes know this leader. The algorithm of [74] does not work for dynamic networks. In another work [72], the authors use a spectral technique to "blacklist" malicious nodes leading to faster and more efficient Byzantine agreement in *static* networks. The idea of blacklisting, unfortunately, won't work in the DNC model since the adversary can change the identities of

Byzantine nodes by churning out old ones and introducing new ones. Other works on handling Byzantine nodes in the context of P2P networks include [102, 19, 54, 56, 21, 32, 110]. The work of [62] 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.

6 Summary and Open Problems

Large-scale, highly dynamic networks are increasingly dominant in the real world. Distributed algorithms that are robust, efficient, and secure are required. An important goal is to solve fundamental distributed computing problems with provable guarantees, under strong models. This paper presents models, algorithms, and techniques that work even under a high amount of adversarial dynamism and presence of Byzantine nodes. The following are main highlights of the results discussed:

- The DNC model is a model for dynamic networks where both nodes *and* edges change continuously over time (with no stabilization or quiescence) and allows a high adversarial churn rate and presence of significant amount of Byzantine nodes. The model generalizes the *churn-free* dynamic model, where only edges change and nodes remain fixed.

- It is shown that one can efficiently perform non-trivial distributed tasks such as agreement, leader election, and storage and search even under a high rate of adversarial churn and presence of a large number of Byzantine nodes under an oblivious adversary (both the churn and the topology are controlled by an oblivious adversary). All these problems can be solved in polylogarithmic (in n, where, is the stable network size) rounds in a scalable way (i.e., nodes send and receive only polylogarithmic sized messages per round and process only polylogarithmic bits per round) and can tolerate up to $O(\sqrt{n}/\operatorname{polylog}(n))$ churn (per round) and $O(\sqrt{n}/\operatorname{polylog}(n))$ Byzantine nodes (per round). If there is only churn, then one can tolerate up to ϵn churn rate for agreement and $O(n/\operatorname{polylog}(n))$ churn rate for storage and search; again, the protocols take polylogarithmic rounds and are scalable.

- Information spreading, support estimation, random walks, and expansion properties of the underlying graph prove crucial in designing robust, efficient, and local algorithms that tolerate a high degree of churn and Byzantine nodes.

- The results and techniques also seamlessly apply to dynamic models without churn (where only edges change) as well as to static networks (where nodes and edges remain fixed, but some nodes can fail and others can be Byzantine). While the amount of failures and Byzantine nodes that can be tolerated is less compared to protocols that are designed specifically for static networks (where up to a linear fraction of Byzantine nodes can be tolerated [108]), the protocols here are extremely lightweight, simple, and scalable compared to prior protocols.

- A key ingredient for all the results discussed here is that the underlying graph be an expander at all times. To implement this in a distributed manner under high adversarial churn is challenging. The article discusses a protocol that essentially achieves this — it maintains a large expander subgraph in every round with high probability.

There are a number of key problems for further research:

1. Extend the results presented in this paper to general graphs, in particular to those which have expansion less than a constant. More precisely, quantify tradeoffs between the amount of churn that can be tolerated with the properties of the graph such as expansion or conductance.

2. An important open problem is whether we can tolerate significantly more Byzantine nodes in a dynamic network when the churn adversary is *oblivious*. More precisely, can we perform efficient (polylogarithmic rounds) Byzantine agreement and leader election when the number of Byzantine nodes is significantly more than $\sqrt{n}/\operatorname{polylog}(n))$ (per round) under an oblivious adversary that controls both the churn and the topology. A similar question also applies to the storage and search problem under presence of Byzantine nodes. Note that the $O(\sqrt{n}/\operatorname{polylog}(n))$ lower bound on the number of Byzantine nodes (per round) and churn rate applies only when the churn adversary is *adaptive*; in fact, this bound holds even when there are no Byzantine nodes and there is only adaptive churn [12]. The oblivious model is important for various reasons. Historically, oblivious models in the Byzantine setting have been hard to solve (see. e.g., [27]); also it is less stringent and arguably more realistic. It might be possible to tolerate as much as $O(n/\operatorname{polylog}(n))$ churn in this model; on the other hand, no lower bounds are known. Another key open problem is whether one can design a *scalable* (that processes and sends only polylogarithmic bits per node per round) algorithm for the agreement problem (even without Byzantine nodes) under the *adaptive* churn adversary.

3. Is it possible to design protocols that work when the network size is not known and may even change over time? This will result in a protocol that has fully local knowledge (the current protocols need knowledge of n and allows only little change in network size). This issue requires devising a distributed protocol that can measure network parameters such as size, average degree etc. under churn and Byzantine nodes.

References

[1] P2p networks hijacked for ddos attacks. 2007. `http://news.netcraft.com/archives/2007/05/23/p2p_networks_hijacked_for_ddos_attacks.html`.

[2] Sebastian Abshoff, Markus Benter, Manuel Malatyali, and Friedhelm Meyer auf der Heide. On two-party communication through dynamic networks. *OPODIS*, pages 11–22, 2013.

[3] Yehuda Afek, Baruch Awerbuch, and Eli Gafni. Applying static network protocols to dynamic networks. *FOCS'87*, pages 358–370, 1987.

[4] Yehuda Afek, Eli Gafni, and Adi Rosen. The slide mechanism with applications in dynamic networks. *ACM PODC*, pages 35–46, 1992.

[5] Rudolf Ahlswede, Ning Cai, Shuo-Yen Li, and Raymond Yeung. Network information flow. *IEEE Transactions on Information Theory*, 46(4):1204–1216, 2000.

[6] James Aspnes. Lower bounds for distributed coin-flipping and randomized consensus. In *Proceedings of the twenty-ninth annual ACM symposium on Theory of computing (STOC)*, pages 559–568, 1997.

[7] James Aspnes and Gauri Shah. Skip graphs. *SODA*, pages 384–393, 2003.

[8] James Aspnes and Udi Wieder. The expansion and mixing time of skip graphs with applications. *SPAA*, pages 126–134, 2005.

[9] Hagit Attiya and Jennifer Welch. *Distributed Computing: Fundamentals, Simulations and Advanced Topics (2nd edition)*. John Wiley Interscience, March 2004.

[10] John Augustine, Tejas Kulkarni, and Sumathi Sivasubramaniam. Leader election in sparse dynamic networks with churn. *IPDPS*, 347-356, 2015.

[11] John Augustine, Anisur Rahaman Molla, Ehab Morsy, Gopal Pandurangan, Peter Robinson, and Eli Upfal. Storage and search in dynamic peer-to-peer networks. In *SPAA*, pages 53–62, 2013.

[12] John Augustine, Gopal Pandurangan, and Peter Robinson. Fast byzantine agreement in dynamic networks. *PODC*, pages 74–83, 2013.

[13] John Augustine, Gopal Pandurangan, and Peter Robinson. Fast byzantine leader election in dynamic networks. *DISC*, pages 276–291, 2015.

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

[15] John Augustine, Gopal Pandurangan, Peter Robinson, and Eli Upfal. Towards robust and efficient computation in dynamic peer-to-peer networks. *SODA*, pages 551–569, 2012.

[16] Chen Avin, Michal Koucký, and Zvi Lotker. How to explore a fast-changing world (cover time of a simple random walk on evolving graphs). *ICALP (1)*, pages 121–132, 2008.

[17] B. Awerbuch and F. T. Leighton. Improved approximation algorithms for the multi-commodity flow problem and local competitive routing in dynamic networks. *ACM STOC*, pages 487–496, May 1994.

[18] Baruch Awerbuch, Boaz Patt-Shamir, David Peleg, and Michael E. Saks. Adapting to asynchronous dynamic networks. *STOC'92*, pages 557–570, 1992.

[19] Baruch Awerbuch and Christian Scheideler. Group Spreading: A Protocol for Provably Secure Distributed Name Service. *ICALP*, pages 183–195, 2004.

[20] Baruch Awerbuch and Christian Scheideler. The hyperring: a low-congestion deterministic data structure for distributed environments. *SODA*, pages 318–327, 2004.

[21] Baruch Awerbuch and Christian Scheideler. Robust random number generation for peer-to-peer systems. *OPODIS*, pages 275–289, 2006.

[22] Baruch Awerbuch and Christian Scheideler. Towards a scalable and robust DHT. *Theory of Computing Systems*, 45:234–260, 2009.

[23] Baruch Awerbuch and Michael Sipser. Dynamic networks are as fast as static networks (preliminary version). In *29th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 206–220, 1988.

[24] Ozalp Babaoglu, Moreno Merzolla, and Michele Tamburini. Design and implementation of a p2p cloud system. *SAC*, 412-417, 2012.

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

[26] Ziv Bar-Joseph and Michael Ben-Or. A tight lower bound for randomized synchronous consensus. *Proceedings of the seventeenth annual ACM symposium on Principles of distributed computing (PODC)*, pages 193–199, 1998.

[27] M. Ben-Or and N. Linial. Collective coin flipping. In Silvio Micali, editor, *Advances in Computing Research 5: Randomness and Computation*, volume 5, pages 91–115, JAI Press, 1989.

[28] Website of Bitcoin. `http://www.bitcoin.org/`.

[29] Marc Bui, Thibault Bernard, Devan Sohier, and Alain Bui. Random walks in distributed computing: A survey. In *4th International Workshop on Innovative Internet Community Systems (IICS)*, pages 1–14, 2004.

[30] John F. Canny. Collaborative filtering with privacy. In *IEEE Symposium on Security and Privacy*, pages 45–57, 2002.

[31] Arnaud Casteigts, Paola Flocchini, Walter Quattrociocchi, and Nicola Santoro. Time-varying graphs and dynamic networks. *CoRR*, abs/1012.0009, 2010. Short version in ADHOC-NOW 2011.

[32] M. Castro, P. Druschel, A. Ganesh, A. Rowstron, and D. Wallach. Secure Routing for Structured Peer-to-Peer Overlay Networks. In *OSDI*, pages 299–314, 2002.

[33] Yu-Wei Chan, Tsung-Hsuan Ho, Po-Chi Shih, and Yeh-Ching Chung. Malugo: A peer-to-peer storage system. *Int. J. ad hoc and ubiquitous computing*, 5(4), 209-218, 2010.

[34] Nicolas Christin, Andreas S. Weigend, and John Chuang. Content availability, pollution and poisoning in file sharing peer-to-peer networks. In *Proceedings 6th ACM Conference on Electronic Commerce (EC)*, pages 68–77, 2005.

[35] Hyun Chul Chung, Peter Robinson, and Jennifer L. Welch. Regional consecutive leader election in mobile ad-hoc networks. In *DIALM-PODC*, pages 81–90, 2010.

[36] Hyun Chul Chung, Peter Robinson, and Jennifer L. Welch. Optimal regional consecutive leader election in mobile ad-hoc networks. *FOMC*, pages 52–61, 2011.

[37] Andrea Clementi, Riccardo Silvestri, and Luca Trevisan. Information spreading in dynamic graphs. *PODC*, 37-46, 2012.

[38] Website of Cloudmark Inc. http://cloudmark.com/.

[39] Edith Cohen. Size-estimation framework with applications to transitive closure and reachability. *J. Comput. Syst. Sci.*, 55(3):441–453, 1997.

[40] Alejandro Cornejo, Seth Gilbert, and Calvin C. Newport. Aggregation in dynamic networks. In *PODC*, pages 195–204, 2012.

[41] Miguel Correia, Giuliana Santos Veronese, Nuno Ferreira Neves, and Paulo Verissimo. Byzantine consensus in asynchronous message passing systems; a survey. *Int. J. Crit. Comput.-Based Syst.*, 2(2):141–161, July 2011.

[42] Website of Crashplan Inc. http://www.crashplan.com/.

[43] A. Das Sarma, A. Molla, and G. Pandurangan. Fast distributed computation in dynamic networks via random walks. *DISC*, 136-150, 2012. Journal version: *Theoretical Computer Science*, 381, 45-66, 2015.

[44] Souptik Datta, Kanishka Bhaduri, Chris Giannella, Ran Wolff, and Hillol Kargupta. Distributed data mining in peer-to-peer networks. *IEEE Internet Computing*, 10(4):18–26, 2006.

[45] Amir Dembo and Ofer Zeitouni. Large deviations techniques and applications. Springer, 2010.

[46] Benjamin Doerr, Leslie Ann Goldberg, Lorenz Minder, Thomas Sauerwald, and Christian Scheideler. Stabilizing consensus with the power of two choices. In *SPAA*, pages 149–158, 2011.

[47] Danny Dolev and H. Raymond Strong. Authenticated algorithms for byzantine agreement. *SIAM J. Comput.*, 12(4):656–666, 1983.

[48] Shlomi Dolev. *Self-stabilization*. MIT Press, Cambridge, MA, USA, 2000.

[49] P. Druschel and A. Rowstron. Past: A large-scale, persistent peer-to-peer storage utility. In *HotOS VIII*, pages 75–80, 2001.

[50] P. Druschel and A. Rowstron. Storage management and caching in past, a large-scale, persistent peer-to-peer storage utility. *In Proc. of ACM SOSP*, 188-201, 2001.

[51] Chinmoy Dutta, Gopal Pandurangan, Rajmohan Rajaraman, Zhifeng Sun, and Emanuele Viola. On the complexity of information spreading in dynamic networks. *SODA*, pages 717–736, 2013.

[52] Cynthia Dwork, David Peleg, Nicholas Pippenger, and Eli Upfal. Fault tolerance in networks of bounded degree. *SIAM J. Comput.*, 17(5):975–988, 1988.

[53] Jarret Falkner, Michael Piatek, John P. John, Arvind Krishnamurthy, and Thomas E. Anderson. Profiling a million user DHT. In *Internet Measurement Conference*, pages 129–134, 2007.

[54] Amos Fiat, Steve Gribble, Anna Karlin, Jared Saia, and Stefan Saroiu. Dynamically Fault-Tolerant Content Addressable Networks. *Proceedings of the First International Workshop on Peer-to-Peer Systems*, 270-279, 2002.

[55] Amos Fiat and Jared Saia. Censorship resistant peer-to-peer content addressable networks. In *SODA*, pages 94–103, 2002.

[56] Amos Fiat, Jared Saia, and Maxwell Young. Making chord robust to byzantine attacks. In *ESA*, pages 803–814, 2005.

[57] Michael J. Fischer and Nancy A. Lynch. A lower bound for the time to assure interactive consistency. *Inf. Process. Lett.*, 14(4):183–186, 1982.

[58] E. Gafni and B. Bertsekas. Distributed algorithms for generating loop-free routes in networks with frequently changing topology. *IEEE Trans. Comm.*, 29(1):1118, 1981.

[59] Roxana Geambasu, Tadayoshi Kohno, Amit A. Levy, and Henry M. Levy. Vanish: Increasing data privacy with self-destructing data. In *USENIX Security Symposium*, pages 299–316, 2009.

[60] Sukumar Ghosh. *Distributed Systems: An Algorithmic Approach*. CRC Press, 2006.

[61] C.M. Grinstead and J.L. Snell. *Introduction to probability*. American Mathematical Society, 1997.

[62] Rachid Guerraoui, Florian Huc, and Anne-Marie Kermarrec. Highly dynamic distributed computing with byzantine failures. In *PODC*, pages 176–183, 2013.

[63] P. Krishna Gummadi, Stefan Saroiu, and Steven D. Gribble. A measurement study of napster and gnutella as examples of peer-to-peer file sharing systems. *Computer Communication Review*, 32(1):82, 2002.

[64] Bernhard Haeupler and David Karger. Faster information dissemination in dynamic networks via network coding. In *ACM PODC*, pages 381–390, 2011.

[65] Ragib Hasan, Zahid Anwar, William Yurcik, Larry Brumbaugh, and Roy Campbell. A survey of peer-to-peer storage techniques for distributed file systems. In *Proc of ITCC*, pages 205–213, 2005.

[66] Kirsten Hildrum and John Kubiatowicz. Asymptotically efficient approaches to fault-tolerance in peer-to-peer networks. In *DISC*, pages 321–336, 2003.

[67] Riko Jacob, Andréa W. Richa, Christian Scheideler, Stefan Schmid, and Hanjo Täubig. A distributed polylogarithmic time algorithm for self-stabilizing skip graphs. *PODC*, pages 131–140, 2009.

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

[69] Bruce M. Kapron, David Kempe, Valerie King, Jared Saia, and Vishal Sanwalani. Fast asynchronous byzantine agreement and leader election with full information. *ACM Transactions on Algorithms*, 6(4), 2010.

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

[71] Valerie King and Jared Saia. Breaking the $O(n^2)$ bit barrier: Scalable byzantine agreement with an adaptive adversary. *J. ACM*, 58:18:1–18:24, July 2011.

[72] Valerie King and Jared Saia. Faster agreement via a spectral method for detecting malicious behavior. In *SODA*, pages 785–800, 2014.

[73] Valerie King, Jared Saia, Vishal Sanwalani, and Erik Vee. Scalable leader election. In *SODA*, pages 990–999, 2006.

[74] Valerie King, Jared Saia, Vishal Sanwalani, and Erik Vee. Towards secure and scalable computation in peer-to-peer networks. In *FOCS*, pages 87–98, 2006.

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

[76] Fabian Kuhn, Nancy Lynch, and Rotem Oshman. Distributed computation in dynamic networks. In *ACM STOC*, pages 513–522, 2010.

[77] Fabian Kuhn, Rotem Oshman, and Yoram Moses. Coordinated consensus in dynamic networks. PODC, pages 1–10, 2011.

[78] Fabian Kuhn, Stefan Schmid, and Roger Wattenhofer. Towards worst-case churn resistant peer-to-peer systems. *Distributed Computing*, 22(4):249–267, 2010.

[79] Shay Kutten, Gopal Pandurangan, David Peleg, Peter Robinson, and Amitabh Trehan. On the complexity of universal leader election. PODC, pages 100–109, 2013. Journal version: *Journal of the ACM*, 62(1), 7:1-7:27, 2015.

[80] C. Law and K.-Y. Siu. Distributed construction of random expander networks. *INFOCOM 2003*, pages 2133 – 2143, 2003.

[81] James Li. A survey of peer-to-peer network security issues. 2007. `http://www.cse.wustl.edu/~jain/cse571-07/ftp/p2p/`.

[82] Eng Keong Lua, Jon Crowcroft, Marcelo Pias, Ravi Sharma, and Steven Lim. A survey and comparison of peer-to-peer overlay network schemes. *IEEE communications surveys and tutorials*, 7(1-4), 72-93, 2005.

[83] Nancy Lynch. *Distributed Algorithms*. Morgan Kaufman Publishers, San Francisco, USA, 1996.

[84] Peter Mahlmann and Christian Schindelhauer. Peer-to-peer networks based on random transformations of connected regular undirected graphs. *SPAA*, pages 155–164, 2005.

[85] David J. Malan and Michael D. Smith. Host-based detection of worms through peer-to-peer cooperation. *WORM*, pages 72–80, 2005.

[86] Michael Mitzenmacher and Eli Upfal. *Probability and Computing: Randomized Algorithms and Probabilistic Analysis*. Cambridge University Press, 2005.

[87] Damon Mosk-Aoyama and Devavrat Shah. Fast distributed algorithms for computing separable functions. *IEEE Transactions on Information Theory*, 54(7):2997–3007, 2008.

[88] Moni Naor and Udi Wieder. Scalable and dynamic quorum systems. In *PODC*, 114-122, 2003.

[89] Moni Naor and Uri Nadav. The dynamic and-or quorum system. In *DISC*, 472-486, 2005.

[90] Moni Naor and Udi Wieder. A simple fault tolerant distributed hash table. In *IPTPS*, pages 88–97, 2003.

[91] Regina O'Dell and Roger Wattenhofer. Information dissemination in highly dynamic graphs. In *DIALM-POMC*, pages 104–110, 2005.

[92] Gopal Pandurangan, Prabhakar Raghavan, and Eli Upfal. Building low-diameter P2P networks. In *FOCS*, pages 492–499, 2001.

[93] Gopal Pandurangan, Peter Robinson, and Amitabh Trehan. Dex: Self-healing expanders. *IEEE IPDPS*, 702-711, 2014.

[94] Gopal Pandurangan and Amitabh Trehan. Xheal: localized self-healing using expanders. In Cyril Gavoille and Pierre Fraigniaud, editors, *PODC*, pages 301–310, 2011.

[95] Marshall C. Pease, Robert E. Shostak, and Leslie Lamport. Reaching agreement in the presence of faults. *J. ACM*, 27(2):228–234, 1980.

[96] Arjan Peddemors. Cloud storage and peer-to-peer storage - end-user considerations and product overview. *http://www.novay.nl/okb/publications/152*, 2010.

[97] David Peleg. *Distributed computing: a locality-sensitive approach*. Society for Industrial and Applied Mathematics, 2000.

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

[99] 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*, 329–350, 2001.

[100] J. Saia, A. Fiat, S. Gribble, A. Karlin, and S. Saroiu. Dynamically fault-tolerant content addressable networks. *Proceedings of the 1st International Workshop on Peer-to-Peer Systems*, 270-279, 2002.

[101] Atish Das Sarma, Danupon Nanongkai, Gopal Pandurangan, and Prasad Tetali. Distributed random walks. *J. ACM*, 60(1):2, 2013.

[102] Christian Scheideler. How to spread adversarial nodes?: rotate! *STOC*, pages 704–713, 2005.

[103] Christian Scheideler and Stefan Schmid. A distributed and oblivious heap. *ICALP*, pages 571–582. 2009.

[104] Subhabrata Sen and Jia Wang. Analyzing peer-to-peer traffic across large networks, *IMC*, pages 137–150, 2002.

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

[106] Daniel Stutzbach and Reza Rejaie. Understanding churn in peer-to-peer networks. *IMC*, 189-202, 2006.

[107] Website of Symform:. http://www.symform.com/.

[108] Eli Upfal. Tolerating a linear number of faults in networks of bounded degree. *Inf. Comput.*, 115(2):312–320, 1994.

[109] Vasileios Vlachos, Stephanos Androutsellis-Theotokis, and Diomidis Spinellis. Security applications of peer-to-peer networks. *Comput. Netw.*, 45:195–205, June 2004.

[110] Maxwell Young, Aniket Kate, Ian Goldberg, and Martin Karsten. Practical Robust Communication in DHTs Tolerating a Byzantine Adversary. In *ICDCS*, pages 263–272, 2010.

[111] 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.

[112] Ming Zhong and Kai Shen. Random walk based node sampling in self-organizing networks. *Operating Systems Review*, 40(3):49–55, 2006.