

Distributed Agreement in Dynamic Peer-to-Peer Networks*

John Augustine[†] Gopal Pandurangan[‡] Peter Robinson[§] Eli Upfal[¶]

Abstract

Motivated by the need for robust and fast distributed computation in highly dynamic Peer-to-Peer (P2P) networks, we study algorithms for the fundamental distributed agreement problem. P2P networks are highly dynamic networks that experience heavy node *churn* (i.e., nodes join and leave the network continuously over time). Our goal is to design fast algorithms (running in a small number of rounds) that guarantee, despite high node churn rate, that almost all nodes reach a stable agreement. Our main contributions are randomized distributed algorithms that guarantee *stable almost-everywhere agreement* with high probability even under high adversarial churn in a polylogarithmic number of rounds. In particular, we present the following results:

1. An $O(\log^2 n)$ -round (n is the stable network size) randomized algorithm that achieves almost-everywhere agreement with high probability under up to *linear churn per round* (i.e., εn , for some small constant $\varepsilon > 0$), assuming that the churn is controlled by an oblivious adversary (that has complete knowledge and control of what nodes join and leave and at what time and has unlimited computational power, but is oblivious to the random choices made by the algorithm). Our algorithm requires only polylogarithmic in n bits to be processed and sent (per round) by each node.
2. An $O(\log m \log^3 n)$ -round randomized algorithm that achieves almost-everywhere agreement with high probability under up to $\varepsilon\sqrt{n}$ churn per round (for some small $\varepsilon > 0$), where m is the size of the input value domain, that works even under an adaptive adversary (that also knows the past random choices made by the algorithm). This algorithm requires up to polynomial in n bits (and up to $O(\log m)$ bits) to be processed and sent (per round) by each node.

Our algorithms are the first-known, fully-distributed, agreement algorithms that work under highly dynamic settings (i.e., high churn rates per step). Furthermore, they are localized (i.e., do not require any global topological knowledge), simple, and easy to implement. These algorithms can serve as building blocks for implementing other non-trivial distributed computing tasks in dynamic P2P networks.

*A preliminary version of this paper appeared in the Proceedings of the ACM/SIAM Symposium on Discrete Algorithms (SODA), 2012, 551-569.

[†]Department of Computer Science and Engineering, Indian Institute of Technology Madras, Chennai, India. E-mail: augustine@cse.iitm.ac.in. Work done while at the Division of Mathematical Sciences, Nanyang Technological University, Singapore 637371.

[‡]Division of Mathematical Sciences, Nanyang Technological University, Singapore 637371 and Department of Computer Science, Brown University, Box 1910, Providence, RI 02912, USA. E-mail: gopalpandurangan@gmail.com. Work supported in part by the following grants: Nanyang Technological University grant M58110000, Singapore Ministry of Education (MOE) Academic Research Fund (AcRF) Tier 2 grant MOE2010-T2-2-082, US NSF grant CCF-1023166, and a grant from the US-Israel Binational Science Foundation (BSF).

[§]Department of Computer Science, National University of Singapore. E-mail: robinson@comp.nus.edu.sg

[¶]Department of Computer Science, Brown University, Box 1910, Providence, RI 02912, USA. E-mail: eli@cs.brown.edu

1 Introduction

Peer-to-peer (P2P) computing is emerging as one of the key networking technologies in recent years with many application systems, e.g., Skype, BitTorrent, Cloudmark etc. However, many of these systems are not truly P2P, as they are not fully decentralized — they typically use hybrid P2P along with centralized intervention. For example, Cloudmark [25] 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 algorithms for large-scale dynamic P2P 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 [33, 40, 64, 65] show that the churn rate 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*.) We note that peer-to-peer algorithms have been proposed for a wide variety of computationally challenging tasks such as collaborative filtering [19], spam detection [25], data mining [28], worm detection and suppression [55, 67], and privacy protection of archived data [38]. However, all algorithms proposed for these problems have no theoretical guarantees of being able to work in a network with a dynamically changing topology and a linear churn rate per round. This is a major bottleneck in implementation and wide-spread use of these algorithms.

In this paper, we take a step towards designing robust algorithms for large-scale dynamic peer-to-peer networks. In particular, we study the fundamental distributed agreement problem in P2P networks (the formal problem statement and model is given in Section 2). An efficient solution to the agreement problem can be used as a building block for robust and efficient solutions to other problems as mentioned above. However, the distributed agreement problem in P2P networks is challenging since the goal is to guarantee *almost-everywhere* agreement, i.e., almost all nodes¹ should reach consensus, even under high churn rate. The churn rate can be as much as linear *per time step (round)*, i.e., up to a constant fraction of the stable network size can be replaced per time step. Indeed, until recently, almost all the work known in the literature (see e.g., [32, 44, 45, 46, 66]) have addressed the almost-everywhere agreement problem 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 Upfal [66] showed how one can achieve almost-everywhere agreement under up to a *linear* number — up to εn , for a sufficiently small $\varepsilon > 0$ — of Byzantine faults in a bounded-degree expander network (n is the network size). The algorithm required $O(\log n)$ rounds and polynomial (in n) number 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 King et al. [47] is important in the context of P2P networks, as it was the first to study scalable (polylogarithmic communication and number of rounds) algorithms for distributed agreement (and leader election) that are tolerant to Byzantine faults. However, as pointed out by the authors, their algorithm works only for static networks; similar to Upfal’s algorithm, the nodes require hardcoded information on

¹In 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 [32].

the network topology to begin with and thus the algorithm does not work when the topology changes. In fact, this work ([47]) raises the open question of whether one can design agreement protocols that can work in highly dynamic networks with a large churn rate.

1.1 Our Main Results

Our first contribution is a rigorous theoretical framework for the design and analysis of algorithms for highly dynamic distributed systems with churn. We briefly describe the key ingredients of our model here. (Our model is described in detail in Section 2.) Essentially, we model a P2P network as a bounded-degree expander graph whose topology — both nodes and edges — can change arbitrarily from round to round and is controlled by an adversary. However, we assume that the total number of nodes in the network is stable. The number of node changes *per round* is called the *churn rate* or *churn limit*. We consider a churn rate of up to some εn , where n is the stable network size. Note that our model is quite general in the sense that we only assume that the topology is an expander at every step; no other special properties are assumed. Indeed, expanders have been used extensively to model dynamic P2P networks² in which the expander property is preserved under insertions and deletions of nodes (e.g., [52, 59]). Since we do not make assumptions on how the topology is preserved, our model is applicable to all such expander-based networks. (We note that various prior work 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 1.3. 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 [20].)

We study stable, almost-everywhere, agreement in our model. By “almost-everywhere”, we mean that almost all nodes, except possibly $\beta c(n)$ nodes (where $c(n)$ is the order of the churn and $\beta > 0$ is a suitably small constant — cf. Section 2) should reach agreement on a common value. (This agreed value must be the input value of some node.) By “stable” we mean that the agreed value is preserved subsequently after the agreement is reached.

Our main contribution is the design and analysis of randomized distributed algorithms that guarantee stable almost-everywhere agreement with high probability (i.e., with probability $1 - 1/n^\gamma$, for an arbitrary fixed constant $\gamma \geq 1$) even under high adversarial churn in a polylogarithmic number of rounds. Our algorithms also guarantee stability once agreement has been reached. In particular, we present the following results (the precise theorem statements are given in the respective sections below):

1. (cf. Section 4) An $O(\log^2 n)$ -round (n is the stable network size) randomized algorithm that achieves almost-everywhere agreement with high probability under up to *linear* churn *per round* (i.e., εn , for some small constant $\varepsilon > 0$), assuming that the churn is controlled by an oblivious adversary (that has complete knowledge of what nodes join and leave and at what time, but is oblivious to the random choices made by the algorithm). Our algorithm requires only polylogarithmic in n bits to be processed and sent (per round) by each node.
2. (cf. Section 5) An $O(\log m \log^3 n)$ -round randomized algorithm that achieves almost-everywhere agreement with high probability under up to $\varepsilon \sqrt{n}$ churn *per round*, for some small $\varepsilon > 0$,

²Expander 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 [32, 44, 45, 46, 66]). Here we show that similar expansion properties are beneficial in the more challenging setting of dynamic networks.

that works even under an adaptive adversary (that also knows the past random choices made by the algorithm). Here m refers to the size of the domain of input values. This algorithm requires up to polynomial in n bits (and up to $O(\log m)$ bits) to be processed and sent (per round) by each node.

3. (cf. Section 6) We also show that no deterministic algorithm can guarantee almost-everywhere agreement (regardless of the number of rounds), even under constant churn rate.

To the best of our knowledge, our algorithms are the first-known, fully-distributed, agreement algorithms that work under highly dynamic settings. Our algorithms are localized (do not require any global topological knowledge), simple, and easy to implement. These algorithms can serve as building blocks for implementing other non-trivial distributed computing tasks in P2P networks.

1.2 Technical Contributions

The main technical challenge that we have to overcome is designing and analyzing distributed algorithms in networks where both nodes and edges can change by a large amount. Indeed, when the churn rate is linear, i.e., say εn per round, in constant $(1/\varepsilon)$ number of rounds the entire network can be renewed!

We derive techniques for information spreading (cf. Section 3) for doing non-trivial distributed computation in such networks. The first technique that we use is flooding. We show that in an expander-based P2P 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) nodes initiate the information spreading (cf. Lemma 3.1). In other words, even an adaptive adversary cannot “suppress” more than a small fraction of the values. The precise statements and proofs are in Section 3.

To analyze these flooding techniques we introduce the dynamic distance, which describes the effective distance between two nodes with respect to the causal influence. We define the notions of influence sets and dynamic distance (or flooding time) in dynamic networks with node churn. (Similar notions have been defined for dynamic graphs with a fixed set of nodes, e.g., [48, 17]). In (connected) networks where the nodes are fixed, the effective diameter (e.g., [48]) is always finite. In the highly dynamic setting considered here, however, the effective distance between two nodes might be infinite, thus we need a more refined definition for influence set and dynamic distance.

The second technique that we use is “support estimation” (cf. Section 3.4). Support estimation is a randomized technique that allows us 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 [26, 56]). Support estimation allows us to estimate the aggregate value quite precisely with high probability even under linear churn. 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.

Apart from support estimation, we also use our flooding techniques in the agreement algorithm for the oblivious case (cf. Algorithm 2) to sway the decision one way or the other. For the adaptive case (cf. Algorithm 3), we use the variance property of a certain probability distribution to achieve the same effect with constant probability.

1.3 Other Related Work

1.3.1 Distributed 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., [7, 53] and the references therein). The relaxation of achieving agreement “almost everywhere” was introduced by [32] 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 [66], which showed how to guarantee almost everywhere agreement in the presence of a linear fraction of faulty nodes. Both the work of [32, 66] crucially use expander graphs to show their results. We also refer to the related results of Berman and Garay on the butterfly network [18].

1.3.2 Byzantine Agreement

We note that Byzantine adversaries are quite different from the adversaries considered in this paper. A Byzantine adversary can have nodes behaving arbitrarily, but no new nodes are added (i.e., no churn), whereas in our 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 [35, 42, 57, 62]. 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 [47] in the context of the leader election problem; essentially, [47] is a sparse (expander) network implementation of the full information protocol of [46]. More specifically, [47] 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 [47] 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. Again, we note that the failure assumption of [47] is quite different from the one we use: Even though we do not assume corrupted nodes, the adversary is free to subject different nodes to churn in every round. Also note that the algorithm of [47] does not work for dynamic networks.

Other works on handling Byzantine nodes in the context of P2P networks include [62, 12, 34, 36, 14, 21, 68].

In [8], we have developed an almost-everywhere agreement algorithm that tolerates up to $\tilde{O}(\sqrt{n})$ churn and $\tilde{O}(\sqrt{n})$ churn per round, in a dynamic network model.

1.3.3 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 [7, 53]). 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 [1, 31, 37]). Some of the early work on general dynamic networks include [2, 11] which introduce general building blocks for communication protocols on dynamic networks. Another

notable work is the local balancing approach of [10] 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 [9, 27, 58, 50] and others; see the recent survey of [20] and the references therein. The works of [50, 9, 27] 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* ([50] and [27] 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 [50]. The recent work of [24], studies the flooding time of *Markovian* evolving dynamic graphs, a special class of evolving graphs.

We note that the model of [49] allows only edge changes from round to round while the nodes remain fixed. In this work, we introduce a dynamic network model where both nodes and edges can change by a large amount (up to a linear fraction of the network size). Therefore, the framework we introduce in Section 2 is more general than the model of [49], as it is additionally applicable to dynamic settings with node churn. The same is true for the notions of dynamic distance and influence set that we introduce in Section 3.1, since in our model the dynamic distance is not necessarily finite. In fact, according to [48], coping with churn is one of the important open problems in the context of dynamic networks. Our paper takes a step in this direction.

An important aspect of our algorithms is that they will work and terminate correctly even when the network keeps continually changing. We note that there has been considerable prior work in dynamic P2P networks (see [59] and the references therein) but these do not assume that the network keeps continually changing over time.

Due to the mobility of nodes, mobile ad-hoc networks can also be considered as dynamic networks. The focus of [58] 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 [23]. 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 our model, this assumption prohibits the adversary from permanently isolating parts of the network. The recent work of [41] presents information spreading algorithms on dynamic networks based on network coding [3].

1.3.4 Fault-Tolerance

In most work on fault-tolerant agreement problems the adversary a priori commits to a fixed set of faulty nodes. In contrast, [30] 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 [30] 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. The median rule, however, only guarantees that this agreement lasts for some polynomial number of rounds, whereas we are able to retain agreement ad infinitum.

Expander graphs and spectral properties have already been applied extensively to improve the

network design and fault-tolerance in distributed computing (cf. [66, 32, 16]). Law and Siu [52] provide a distributed algorithm for maintaining an expander in the presence of churn with high probability by using Hamiltonian cycles. In [61] 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, [60] 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 [6], it is shown that a SKIP graph (cf. [5]) 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, [43] 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 [13] 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 [54] 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 [54] is only guaranteed with high probability however, assuming that the node degree is $\Omega(\log n)$.

Information spreading in distributed networks is the focus of [22] where it is shown that this problem requires $O(\log n)$ rounds in graphs with a certain conductance in the push/pull model where a node can communicate with a randomly chosen neighbor in every round.

Aspnes et al. [4] consider information spreading via expander graphs against an adversary, which is related to the flooding techniques we derive in Section 3. More specifically, in [4] there are two opposing parties “the alert” and “the worm” (controlled by the adversary) that both try to gain control of the network. In every round each alerted node can alert a constant number of its neighbors, whereas each of the worm nodes can infect a constant number of non-alerted nodes in the network. In [4], Aspnes et al. show that there is a simple strategy to prevent all but a small fraction of nodes from becoming infected and, in case that the network has poor expansion, the worm will infect almost all nodes.

The work of [16] shows that, given a network that is initially an expander and assuming some linear fraction of faults, the remaining network will still contain a large component with good expansion. These results are not directly applicable to dynamic networks with large amount of churn like the ones we are considering, as the topology might be changing and linear churn per round essentially corresponds to $O(n \log n)$ total churn after $\Theta(\log n)$ rounds—the minimum amount of time necessary to solve any non-trivial task in our model.

In the context of maintaining properties in P2P networks, Kuhn et al. consider in [51] 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 [51] 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 [63] how to maintain a distributed heap that allows join and leave operations and, in addition, is resistant to Sybil attacks. A robust distributed implementation of a distributed hash table (DHT) in a P2P network is given by [15], which can withstand two important kind of attacks: adaptive join-leave attacks and adaptive insert/lookup attacks by up to εn adversarial peers. Note that, however, that collisions are likely to occur once the number of attacks becomes $\Omega(\sqrt{n})$.

2 Model and Problem Statement

We are interested in establishing stable agreement in a dynamic peer-to-peer network in which the nodes and the edges change over time. The computation is structured into synchronous rounds, i.e., we assume that nodes run at the same processing speed and any message that is sent by some node u to its (current) neighbors in some round $r \geq 1$ will be received by the end of r . To ensure scalability, we restrict the number of bits sent per round by each node to be polylogarithmic in the size of the input value domain (cf. Section 2.1). For dealing with the much more powerful adaptive adversary, we relax this requirement in Sections 3.5 and 5. We model dynamism in the network as a family of undirected graphs $(G^r)_{r \geq 0}$. At the beginning of each round r we start with the network topology G^{r-1} . Then, the adversary gets to change the network from G^{r-1} to G^r (in accordance to rules outlined below). As is typical, an edge $(u, v) \in E^r$ indicates that u and v can communicate in round r by passing messages. For the sake of readability, we use $V^{[r, r+t]}$ as a shorthand for $\bigcap_{i=r}^{r+t} V^i$. Each node u has a unique identifier and is *churned in* at some round r_i and *churned out* at some $r_o > r_i$. More precisely, for each node u , there is a maximal range $[r_i, r_o - 1]$ such that $u \in V^{[r_i, r_o - 1]}$ and for every $r \notin [r_i, r_o - 1]$, $u \notin V^r$. Any information about the network at large is only learned through the messages that u receives. It has no a priori knowledge about who its neighbors will be in the future. Neither does u know when (or whether) it will be churned out. Note that we do not assume that nodes have access to perfect clocks, but we show (cf. Section 3.3) how the nodes can synchronize their clocks.

We make the following assumptions about the kind of changes that our dynamic network can encounter:

Stable Network Size: For all r , $|V^r| = n$, where n is a suitably large positive integer. This assumption simplifies our analysis. Our algorithms will work correctly as long as the number of nodes is reasonably stable (say, between $n - \kappa n$ and $n + \kappa n$ for some suitably small constant κ). Also, we assume that n (or a constant factor estimate of n) is common knowledge among the nodes in the network³.

Churn: For each $r > 1$,

$$|V^r \setminus V^{r-1}| = |V^{r-1} \setminus V^r| \leq \mathcal{L} = \varepsilon c(n),$$

where \mathcal{L} is the *churn limit*, which is some fixed $\varepsilon > 0$ fraction of the *order of the churn* $c(n)$; the equality in the above equation ensures that the network size remains stable. Our work is aimed at high levels of churn up to a churn limit \mathcal{L} that is linear in n , i.e., $c(n) = n$.

Bounded Degree Expanders: The sequence of graphs $(G^r)_{r \geq 0}$ is an expander family with a vertex expansion of at least α , which is a fixed positive constant.⁴ In other words, the adversary must ensure that for every G^r and every $S \subset V^r$ such that $|S| \leq n/2$, the number of nodes in $V^r \setminus S$ with a neighbor in S is at least $\alpha|S|$. Note that we do not explicitly consider the costs (communication and computation) of maintaining an expander under churn. Instead, we assume that the duration of each time step in our model are normalized to be large enough to encompass an expander maintenance protocol such as [52, 60].

³This assumption is important; estimating n accurately in our model is an interesting problem in itself.

⁴Note that the value of α determines ε , i.e. the fraction of churn that we can tolerate. In particular, to tolerate linear amount of churn, we require constant expansion. In principle, our results can potentially be extended to graphs with weaker expansion guarantees as well; however the amount of churn that can be tolerated will be reduced.

A run of a distributed algorithm consists of an infinite number of rounds. We assume that the following events occur (in order) in every round r :

1. A set of at most \mathcal{L} nodes are churned in and another set of \mathcal{L} nodes are churned out. The edges of G^{r-1} may be changed as well, but G^r has to have a vertex expansion of at least α . These changes are under the control of the adversary.
2. The nodes broadcast messages to their (current) neighbors.
3. Nodes receive messages broadcast by their neighbors.
4. Nodes perform computation that can change their state and determine which messages to send in round $r + 1$.

Bounds on Parameters

Recall that the churn limit $\mathcal{L} = \varepsilon c(n)$, where $\varepsilon > 0$ is a constant and $c(n)$ is the churn order. When $c(n) = n$, ε is the fraction of the nodes churned out/in and therefore we require ε to be less than 1 and must adhere to Equation (1). Moreover, we require the bound $\beta < \frac{1}{12}$ regarding the right hand side of (1). However, when $c(n) \in o(n)$, ε can exceed 1. In the remainder of this paper, we consider β to be a small constant independent of n , such that

$$(1) \quad \frac{\varepsilon(1 + \alpha)}{\alpha} < \beta.$$

It will become apparent in Section 3 that (1) presents a sufficient condition for preventing the adversary from containing the information propagated by a set of $\beta c(n)$ nodes.

and that the *churn expansion ratio* $\frac{\varepsilon(1+\alpha)}{\alpha}$ presents a sufficient condition for information propagation in our model (cf. Lemma 3.1). Finally, we assume that n is suitably large (cf. Equations 7 and 8).

2.1 Stable Agreement

We now define the ALMOST EVERYWHERE STABLE AGREEMENT problem (or just the STABLE AGREEMENT problem for brevity). Each node $v \in V^0$ has 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*. STABLE 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* STABLE AGREEMENT *in* R *rounds*, if it exhibits the following characteristics in every run, for any fixed β adhering to (1).

Validity: If, in some round r , node $u \in V^r$ decides on a value val , then $val \in \mathcal{V}$.

Almost Everywhere Agreement: We say that *the network has reached strong almost everywhere agreement by round* R , if at least $n - \beta c(n)$ nodes in V^R have decided on the same value $val^* \in \mathcal{V}$ and every other node remains undecided, i.e., its decision value is \perp . In particular, no node ever decides on a value $val' \in \mathcal{V}$ in the same run, for $val' \neq val^*$.

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 $r \geq R$, at least $n - \beta c(n)$ nodes in V^r have decided on VAL^* .

We also consider a weaker variant of the above problem that we call **ALMOST EVERYWHERE BINARY CONSENSUS** (or simply, **BINARY CONSENSUS**) where the input values in \mathcal{V} are restricted to $\{0, 1\}$.

We consider two types of adversaries for our randomized algorithms. An *oblivious* adversary must commit in advance to the entire sequence of graphs $(G^r)_{r \geq 0}$. In other words, an oblivious adversary must commit independently of the random choices made by the algorithm. We also consider the more powerful *adaptive* adversary that can observe the entire state of the network in every round r (including all the random choices made until round $r - 1$), and then chooses the nodes to be churned out/in and how to change the topology of G^{r+1} .

For the sake of readability, we treat $\log n$ as an integer and omit the necessary ceiling or floor operations if their application is clear from the context.

3 Techniques for Information Spreading

In this section, we first derive and analyze techniques to spread information in the network despite churn. First, we show that the adversary is unable to prevent a sufficiently large set of nodes (of size at least $\beta c(n)$) to propagate their information to almost all other nodes (cf. Lemma 3.1). Building on this result, we analyze the capability of individual nodes to spread their information. We show in Lemma 3.2 and Corollary 3.3 that at most $\beta c(n)$ nodes can be hindered by the adversary. Finally, we show in Lemmas 3.5 and 3.6 that there is a large set of nodes V^* such that all nodes in V^* are able to propagate their information to a large *common* set of nodes.

In Sections 3.4 and 3.5, we describe how to use the previously derived techniques on information spreading to estimate the “support” (i.e. number) of nodes that belong to a specific category (either red or blue). These protocols will form a fundamental building block for our **STABLE AGREEMENT** algorithms.

Due to the high amount of churn and the dynamically changing network, we use message flooding to disseminate and gather information. We now precisely define flooding. Any node can initiate a message for flooding. Messages that need to be flooded have an indicator bit **BFLOOD** set to 1. Each of these messages also contains a terminating condition. The initiating node sends copies of the message to itself and its neighbors. When a node receives a message with **BFLOOD** set to 1, it continues to send copies of that message to itself and its neighbors in subsequent rounds until the terminating condition is satisfied.

3.1 Dynamic Distance and Influence Set

Informally, the dynamic distance from node u to node v is the number of rounds required for a message at u to reach v . We now formally define the notion of *dynamic distance* of a node v from u starting at round r , denoted by $\text{DD}_r(u \rightarrow v)$. When the subscript r is omitted, we assume that $r = 1$.

Suppose node u joins the network at round r_u , and, from round $\max(r_u, r)$ onward, u initiates a message m for flooding whose terminating condition is: $\langle \text{HAS REACHED } v \rangle$. If u is churned out before r , then $\text{DD}_r(u \rightarrow v)$ is undefined. Suppose the first of those flooded messages reaches v in round $r + \Delta r$. Then, $\text{DD}_r(u \rightarrow v) = \Delta r$. Note that this definition allows $\text{DD}_r(u \rightarrow v)$ 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 that absorb the flooded messages and are then removed from the network before they can propagate these messages any further. The influence set of a node u after R rounds starting at round r is given by:

$$\text{INFL}_r(u, R) = \{v \in V^{r+R} : \text{DD}_r(u \rightarrow v) \leq R\}.$$

Note that we require $\text{INFL}_r(u, R) \subseteq V^{r+R}$. 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. Analogously, we define

$$\text{INFL}_r(U, R) = \cup_{u \in U} \text{INFL}_r(u, R),$$

for any set of nodes $U \subseteq V^r$.

If we consider only a single node u , an (adaptive) adversary can easily prevent the influence set of this node from ever reaching any significant size by simply shielding u with churn nodes that are replaced in every round.⁵

3.2 Properties of Influence Sets

We now focus our efforts on characterizing influence sets. This will help us in understanding how we can use flooding to spread information in the network. For the most part of this section we assume that the network is controlled by an adaptive adversary (cf. Section 2.1). The following lemma shows that the number of nodes that are sufficient to influence almost all the nodes in the network is given by the churn-expansion ratio (cf. Equation (1)):

Lemma 3.1. *Suppose that the adversary is adaptive. Consider any set $U \subseteq V^{r-1}$ (for any $r \geq 1$) such that $|U| \geq \beta c(n)$. Then, after*

$$T = 2 \left\lceil \frac{\log n - \log c(n) - \log(\beta - \frac{\varepsilon(1+\alpha)}{\alpha}) - 1}{\log(1 + \alpha)} \right\rceil$$

number of rounds, it holds that

$$(2) \quad |\text{INFL}_r(U, T)| > n - \beta c(n).$$

When considering linear churn, i.e., $c(n) = n$, the bound T becomes a constant independent of n . On the other hand, when considering a churn order of \sqrt{n} , we get $T \in O(\log n)$.

Proof. Our proof assumes that $r = 1$ for simplicity as the arguments extend quite easily to arbitrary values of r . We proceed in two parts: First we show that the nodes in U influence at least $n/2$ nodes in some T_1 rounds. More precisely, we show that $|\text{INFL}(U, T_1)| \geq n/2$. We use vertex expansion in a straightforward manner to establish this part. Then, in the second part we show that nodes in $\text{INFL}(U, T_1)$ go on to influence more than $n - \beta c(n)$ nodes. We cannot use the vertex expansion in a straightforward manner in the second part because the cardinality of the set that is expanding in influence is larger than $n/2$. Rather, we use a slightly more subtle argument in which we use

⁵An oblivious adversary can achieve the same effect with constant probability for linear churn.

vertex expansion going backward in time. The second part requires another T_1 rounds. Therefore, the two parts together complete the proof when we set $T = 2T_1$.

To begin the first part, consider $U \subseteq V^0$ at the start of round 1 with $|U| \geq \beta c(n)$. In round 1, up to $\varepsilon c(n)$ nodes in U can be churned out. Subsequently, the remaining nodes in U influence some nodes outside U as G^1 is an expander with vertex expansion at least α . More precisely, we can say that

$$(3) \quad |\text{INFL}(U, 1)| \geq (\beta c(n) - \varepsilon c(n))(1 + \alpha).$$

At the start of round 2, the graph changes dynamically to G^2 . In particular, up to $\varepsilon c(n)$ nodes might be churned out and they may all be in $\text{INFL}(U, 1)$ in the worst case. However, the influenced set will again expand. Therefore, $|\text{INFL}(U, 2)|$ cannot be less than $(|\text{INFL}(U, 1)| - \varepsilon c(n))(1 + \alpha) \geq \beta c(n)(1 + \alpha)^2 - \varepsilon c(n)(1 + \alpha)^2 - \varepsilon c(n)(1 + \alpha)$. Of course, there will be more churn at the start of round 3 followed by expansion leading to:

$$\begin{aligned} |\text{INFL}(U, 3)| &\geq \left(\beta c(n)(1 + \alpha)^2 - \varepsilon c(n)(1 + \alpha)^2 \right. \\ &\quad \left. - \varepsilon c(n)(1 + \alpha) \right. \\ &\quad \left. - \varepsilon c(n) \right)(1 + \alpha) \\ &= \beta c(n)(1 + \alpha)^3 - \varepsilon c(n) \sum_{k=1}^3 (1 + \alpha)^k. \end{aligned}$$

This cycle of churn followed by expansion continues and we get the following bound at the end of some round i :

$$\begin{aligned} |\text{INFL}(U, i)| &\geq \beta c(n)(1 + \alpha)^i - \varepsilon c(n) \sum_{k=1}^i (1 + \alpha)^k \\ &= \beta c(n)(1 + \alpha)^i \\ &\quad + \varepsilon c(n) \frac{1 - (1 + \alpha)^{i+1}}{\alpha} - \varepsilon c(n) \end{aligned}$$

Therefore, after

$$(4) \quad T_1 = \left\lceil \frac{\log n - \log c(n) - \log(\beta - \frac{\varepsilon(1+\alpha)}{\alpha}) - 1}{\log(1 + \alpha)} \right\rceil$$

rounds, we get

$$(5) \quad |\text{INFL}(U, T_1)| \geq n/2.$$

Now we move on to the second part of the proof. Let $T = 2T_1$. If $|\text{INFL}(U, T)| > n - \beta c(n)$, we are done. Therefore, for the sake of a contradiction, assume that $|\text{INFL}(U, T)| \leq n - \beta c(n)$. Let $S = V^T \setminus \text{INFL}(U, T)$, i.e., S is the set of nodes in V^T that were not influenced by U at (or before) round T . Moreover, $|S| \geq \beta c(n)$ because we have assumed that $|\text{INFL}(U, T)| \leq n - \beta c(n)$. We will start at round T and work our way backward. For $q \leq T$, let $S^q \subseteq V^q$, be the set of all vertices in V^q that, starting from round q , influenced some vertex in S at or before round T . More precisely,

$$S^q = \{s \in V^q : \text{INFL}_q(s, T - q) \cap S \neq \emptyset\}.$$

Suppose that $|S^{T_1}| > n/2$. Then

$$S^{T_1} \cap \text{INFL}(U, T_1) \neq \emptyset,$$

since $|\text{INFL}(U, T_1)| \geq n/2$ by (5). Consider a node $s^* \in S^{T_1} \cap \text{INFL}(U, T_1)$. Note that s^* was influenced by U and went on to influence some node in S before (or at) round T . However, by definition, no node in S can be influenced by any node in U at or before round T . We have thus reached a contradiction.

We are left with showing that $|S^{T_1}| > n/2$. We start with S and work our way backwards. We know that $|S| \geq \beta c(n) > \beta c(n) - \varepsilon c(n)$. We want to compute the cardinality of S^{T-1} . We first focus on an intermediate set S' , which we define as

$$S' = S \cup \{s' : \exists(s, s') \in E^T\}.$$

Since G^T is an expander, $|S'| \geq |S|(1 + \alpha)$. Furthermore, it is also clear that each node in S' could influence some node in S . Notice that $S' \setminus S^{T-1}$ is the set of nodes in S' that were churned in only at the start of round T . Therefore,

$$\begin{aligned} |S^{T-1}| &\geq |S'| - \varepsilon c(n) \\ &\geq |S|(1 + \alpha) - \varepsilon c(n) \\ &> (\beta c(n) - \varepsilon c(n))(1 + \alpha) - \varepsilon c(n) \\ &= \beta c(n)(1 + \alpha) - \varepsilon c(n)(1 + \alpha) - \varepsilon c(n). \end{aligned}$$

Continuing to work our way backwards in time, we get

$$\begin{aligned} |S^{T-2}| &> \beta c(n)(1 + \alpha)^2 - \varepsilon c(n)(1 + \alpha)^2 \\ &\quad - \varepsilon c(n)(1 + \alpha) - \varepsilon c(n), \end{aligned}$$

Or more generally,

$$\begin{aligned} |S^{T-i}| &> \beta c(n)(1 + \alpha)^i - \varepsilon c(n) \sum_{0 \leq j \leq i} (1 + \alpha)^j \\ &= \beta c(n)(1 + \alpha)^i + \varepsilon c(n) \frac{1 - (1 + \alpha)^{i+1}}{\alpha} \\ &= \beta c(n)(1 + \alpha)^i - \frac{\varepsilon c(n)(1 + \alpha)^{i+1}}{\alpha} + \frac{\varepsilon c(n)}{\alpha}. \end{aligned}$$

We now want the value of i for which

$$|S^{T-i}| > n/2 + \frac{\varepsilon c(n)}{\alpha} > n/2.$$

In other words, we want a value of i such that

$$\beta c(n)(1 + \alpha)^i - \frac{\varepsilon c(n)(1 + \alpha)^{i+1}}{\alpha} + \frac{\varepsilon c(n)}{\alpha} > n/2 + \frac{\varepsilon c(n)}{\alpha},$$

which is obtained when $i = T_1$. Therefore, it is easy to see that if we set $T = 2T_1$, we get $|S^{T_1}| > n/2$, thereby completing the proof. \square \square

At first glance, it might appear to be counterintuitive that the order of the bound T decreases with increasing churn. When the adversary has the benefit of churn that is linear in n , our bound on T is a constant, but when the adversary is limited to a churn order of \sqrt{n} , we get $T \in O(\log n)$. This, however, turns out to be fairly natural when we note that the size of the set U of nodes that we start out with is in proportion to the churn limit.

We say that a node $u \in V^r$ is *suppressed for R rounds* or *shielded by churn* if $|\text{INFL}_r(u, R)| < n - \beta c(n)$; otherwise we say it is *unsuppressed*. The following lemma shows that given a set with cardinality at least $\beta c(n)$ some node in that set will be unsuppressed.

Lemma 3.2. *Consider the adaptive adversary. Let U be any subset of V^{r-1} , $r \geq 1$, such that $|U| \geq \beta c(n)$. Let T be the bound derived in Lemma 3.1. There is at least one $u^* \in U$ such that for some $R \in O(T \log n)$, u^* is unsuppressed, i.e.,*

$$|\text{INFL}_r(u^*, R)| > n - \beta c(n).$$

In particular, when the order of the churn is n , T becomes a constant, and we have $R = O(\log n)$.

Before we proceed with our key arguments of the proof, we state a property of bipartite graphs that we will use subsequently.

Property 1. *Let $H = (A, B, E)$ be a bipartite graph in which $|A| > 1$ and every vertex $b \in B$ has at least one neighbor in A . There is a subset $A^* \subset A$ of cardinality at most $\lceil |A|/2 \rceil$ such that*

$$|\{b : \exists a^* \in A^* \text{ such that } (a^*, b) \in E\}| \geq \lceil |B|/2 \rceil.$$

Proof. (of Property 1) Consider each node in A to be a unique color. Color each node in B using the color of a neighbor in A chosen arbitrarily. Now partition B into maximal subsets of nodes with like colors. Consider the parts of the partition sorted in decreasing order of their cardinalities. We now greedily choose the first $\lceil |A|/2 \rceil$ colors in the sorted order of parts of B . We call the chosen colors C . Observe that colors in C cover at least as many nodes in B as those not in C . Suppose the colors in C cover fewer than $\lceil |B|/2 \rceil$ nodes in B . Then the remaining colors will cover $\lceil |B|/2 \rceil$, but that is a contradiction. Therefore, colors in C cover at least $\lceil |B|/2 \rceil$ nodes in B . The nodes in A that have the colors in C are the nodes that comprise A^* , thereby completing our proof. \square \square

Proof. (of Lemma 3.2) Again, our proof assumes $r = 1$ because it generalizes to arbitrary values of r quite easily. From Lemma 3.1, we know that the influence of all nodes in U taken together will reach $n - \beta c(n)$ nodes in T rounds. This does not suffice because we are interested in showing that there is at least one node in V^0 that (individually) influences $n - \beta c(n)$ nodes in V^R for some $R = O(T \log n)$.

From Lemma 3.1, we know that U (collectively) will influence at least $n - \beta c(n)$ nodes in T rounds, i.e.,

$$|\text{INFL}(U, T)| > n - \beta c(n).$$

From Property 1, we know that there is a set $U_1 \subset U$ of cardinality at most $\lceil |U|/2 \rceil$ such that

$$|\text{INFL}(U_1, T)| > \frac{n - \beta c(n)}{2}.$$

Recalling that $\beta < \frac{1}{12} < \frac{1}{3}$, we know that $|\text{INFL}(U_1, T)| \geq \beta c(n)$. We can again use Lemma 3.1 to say that $\text{INFL}(U_1, T)$ influences more than $n - \beta c(n)$ nodes in additional T rounds and, by transitivity, U_1

influences more than $n - \beta c(n)$ nodes after $2T$ rounds. We therefore have $|\text{INFL}(U_1, 2T)| > n - \beta c(n)$. Again, we can choose a set $U_2 \subset U_1$ (using Property 1) that consists of $\lceil |U_1|/2 \rceil$ nodes in U_1 such that $|\text{INFL}(U_2, 2T)| \geq \beta c(n)$. Subsequently applying Lemma 3.1 extends the influence set of U_2 to more than $n - \beta c(n)$ after $3T$ rounds.

In every iteration i of the above argument, the size of the set U_i decreases by a constant fraction until we are left with a single node $u^* \in U$ such that $|\text{INFL}(u^*, O(\log n)T)| > n - \beta c(n)$. \square \square

Can $\beta c(n)$ (or more nodes) be suppressed for any significant number of (say, $\Omega(T \log n)$) rounds? This is in immediate contradiction to Lemma 3.2 because any such suppressed set of nodes must contain an unsuppressed node. This leads us to the following corollary.

Corollary 3.3. *The number of nodes that can be suppressed for $\Omega(T \log n)$ rounds is less than $\beta c(n)$, even if the network is controlled by an adaptive adversary.*

Corollary 3.4. *Consider an oblivious adversary that must commit to the entire sequence of graphs in advance. If we choose a node u uniformly at random from V^0 , with probability at least $1 - \frac{\beta c(n)}{n}$, then u will be unsuppressed, i.e.,*

$$|\text{INFL}(u, \Omega(T \log n))| > n - \beta c(n).$$

Proof. Let $S \subset V^0$ be the set of nodes suppressed for $\Omega(T \log n)$ rounds. Under an oblivious adversary, the node u chosen uniformly at random from V^0 will not be in S with probability $1 - \frac{\beta c(n)}{n}$, and hence, will not be suppressed with that same probability. \square \square

The following two lemmas show that there exists a set V^* of unsuppressed nodes, all of which can influence a large common set of nodes, given enough time.

Lemma 3.5. *Consider a dynamic network under linear churn that is controlled by an adaptive adversary. In some $r \in O(\log n)$ rounds, there is a set of unsuppressed nodes $V^* \subseteq V^0$ of cardinality more than $(1 - \beta)n$ such that*

$$\left| \bigcap_{v \in V^*} \text{INFL}(v, r) \right| > (1 - \beta)n.$$

Proof. Let $V^* \subseteq V^0$ be any set of unsuppressed nodes, i.e., in some $c_0 \log n$ rounds for some constant c_0 , the influence set of each $v \in V^*$ has cardinality more than $(1 - \beta)n$. Note that, however, we cannot guarantee that, for any two vertices v_1 and v_2 in V^* ,

$$|\text{INFL}(v_1, c_0 \log n) \cap \text{INFL}(v_2, c_0 \log n)| > (1 - \beta)n.$$

Assume for simplicity that $|V^*|$ is a power of 2. Consider any pair of vertices $\{v_1, v_2\}$, both members of V^* . Recalling that $\beta < \frac{1}{12} < \frac{1}{3}$, we can say that

$$|\text{INFL}(v_1, c_0 \log n) \cap \text{INFL}(v_2, c_0 \log n)| \geq \beta n.$$

Therefore, considering that the intersected set $\text{INFL}(v_1, c_0 \log n) \cap \text{INFL}(v_2, c_0 \log n)$ of nodes has cardinality at least βn , we can apply Lemma 3.1 leading to $|\text{INFL}(v_1, c_0 \log n + T) \cap \text{INFL}(v_2, c_0 \log n + T)| > (1 - \beta)n$. We can partition V^* into a set S_1 of $\frac{|V^*|}{2}$ pairs such that for each pair, the intersection of influence sets has cardinality more than $(1 - \beta)n$ after $c_0 \log n + T$ rounds. Similarly, we can

construct a set S_2 of quadruples by disjointly pairing the pairs in S_1 . Using a similar argument, we can say that for any $Q \in S_2$,

$$\left| \bigcap_{v \in Q} \text{INFL}(v, c_0 \log n + 2T) \right| > (1 - \beta)n.$$

Progressing analogously, the set $S_{\log |V^*|}$ will equal V^* and we can conclude that

$$\left| \bigcap_{v \in S_{\log |V^*|}} \text{INFL}(v, c_0 \log n + T \log |V^*|) \right| > (1 - \beta)n.$$

Since $|V^*| \leq n$, it holds that $c_0 \log n + T \log |V^*| \in O(\log n)$, thus completing the proof. \square \square

Lemma 3.6. *Suppose that up to $\varepsilon\sqrt{n}$ nodes can be subjected to churn in any round by an adaptive adversary. In some $r \in O(\log^2 n)$ rounds, there is a set of unsuppressed nodes $V^* \subseteq V^0$ of cardinality at least $n - \beta\sqrt{n}$ such that*

$$\left| \bigcap_{v \in V^*} \text{INFL}(v, r) \right| > n - \beta\sqrt{n}.$$

Proof. Since we assume that $c(n) = \sqrt{n}$, the bound T of Lemma 3.1 is in $O(\log n)$. Therefore, by instantiating Corollary 3.3, we know that each of the unsuppressed nodes in V^* (which is of cardinality at least $n - \beta\sqrt{n}$) will influence more than $n - \beta\sqrt{n}$ nodes in $O(\log^2 n)$ time. We can use the same argument as in Lemma 3.5 to show that in $O(\log n)$ rounds, all the unsuppressed nodes have a common influence set of size at least $\Theta(n)$. That common influence set will grow to at least $n - \beta\sqrt{n}$ nodes within another $O(\log^2 n)$ rounds. Thus a total of $O(\log^2 n)$ rounds is sufficient to fulfill the requirements. \square \square

3.3 Maintaining Information in the Network

In a dynamic network with churn limit εn , the entire set of nodes in the network can be churned out and new nodes churned in within $1/\varepsilon$ rounds. How do the new nodes even know what algorithm is running? How do they know how far the algorithm has progressed? To address these basic questions, the network needs to maintain some global information that is not lost as the nodes in the network are churned out. There are two basic pieces of information that need to be maintained so that a new node can join in and participate in the execution of the distributed algorithm:

1. the algorithm that is currently executing, and
2. the number of rounds that have elapsed in the execution of the algorithm. In other words, a global clock has to be maintained.

We assume that the nodes in V^0 are all synchronized in their understanding of what algorithm to execute and the global clock. The nodes in the network continuously flood information on what algorithm is running so that when a new node arrives, unless it is shielded by churn, it receives this information and can start participating in the algorithm. To maintain the clock value, nodes send their current clock value to their immediate neighbors. When a new node receives the clock information from a neighbor, it sets its own clock accordingly. Since nodes are not malicious or faulty, Lemma 3.1 ensures that information is correctly maintained in more than $n - \beta c(n)$ nodes.

3.4 Support Estimation Under an Oblivious Adversary

Suppose we have a dynamic network with \mathcal{R} nodes colored red in V^0 . \mathcal{R} is also called the *support* of red nodes. We want the nodes in the network to estimate \mathcal{R} under an oblivious adversary. We assume that the adversary chooses \mathcal{R} and which \mathcal{R} nodes in V^0 to color red, but it does not know the random choices made by the algorithm. Furthermore, we assume that churn can be linear in n , i.e., $c(n) = n$.

Our algorithm uses random numbers drawn from the exponential distribution, whose probability density function, we recall, is parameterized by λ and given by $f(x) = \lambda \exp(-\lambda x)$ for all $x \geq 0$. Furthermore, we notice that the expected value of a random number drawn from the exponential distribution of parameter λ is $1/\lambda$. We now present two properties of exponential random variables that are crucial to our context. Consider $K \geq 1$ independent random variables Y_1, Y_2, \dots, Y_K , each following the exponential distribution of rate λ .

Property 2 (see [39] for example). *The minimum among all Y_i 's, for $1 \leq i \leq K$, is an exponentially distributed random variable with parameter $K\lambda$.*

The idea behind our algorithm exploits Property 2 in the following manner. If each of the \mathcal{R} red nodes generate an exponentially distributed random number with parameter 1, 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} . To get a more accurate estimation of \mathcal{R} , we exploit the following property that provides us with sharp concentration when the process is repeated a sufficient number of times.

Property 3 (see [56] and pp. 30, 35 of [29]). *Let $X_K = \frac{1}{K} \sum_{i=1}^K Y_i$. Then, for any $\varsigma \in (0, 1/2)$,*

$$Pr \left(\left| X_K - \frac{1}{\lambda} \right| \geq \frac{\varsigma}{\lambda} \right) \leq 2 \exp \left(-\frac{\varsigma^2 K}{3} \right).$$

We now present our algorithm for estimating \mathcal{R} in pseudocode format (assuming $\mathcal{R} \geq n/2$); see Algorithm 1.

Theorem 3.7. *Consider an oblivious adversary and let γ be an arbitrary fixed constant ≥ 1 . Let $\bar{\mathcal{R}} = \max(\mathcal{R}, n - \mathcal{R})$. By executing Algorithm 1 to estimate both \mathcal{R} and $n - \mathcal{R}$, we can 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 $1 - n^{-\gamma}$.*

Proof. Without loss of generality, let $\mathcal{R} \geq n/2$. Out of the \mathcal{R} red nodes up to βn nodes (chosen obliviously) can be suppressed, leaving us with

$$(6) \quad \mathcal{R}' \geq \mathcal{R} - \beta n \geq (1 - 2\beta)\mathcal{R}$$

unsuppressed red nodes (since $\mathcal{R} \geq n/2$). In a slight abuse of notation, we use \mathcal{R} and \mathcal{R}' to denote both the cardinality and the set of red nodes and unsuppressed red nodes, respectively. We define

$$U = \bigcap_{v \in \mathcal{R}'} \text{INFL}(v, t);$$

note that $t = O(\log n)$ and $|U| \geq (1 - \beta)n$ (cf. Lemma 3.5). Let u be some node in U . Let

$$V_u = \{v : v \in \mathcal{R} \wedge u \in \text{INFL}(v, t)\}.$$

Algorithm 1 Algorithm to estimate the support \mathcal{R} of red nodes when $\mathcal{R} \geq n/2$.

The following pseudocode is executed at every node u .

$P \in \Theta(\log n)$ controls the precision of our estimate. Its exact value is worked out in the proof of Theorem 3.7.

At round 1:

- 1: Draw P random numbers $s_1, s_2, \dots, s_i, \dots, s_P$, each from the exponential random distribution with rate 1.
// Each s_i is chosen with a precision that ensures that the smallest possible positive value is at most $\frac{1}{n^{\Theta(1)}}$;
// Note that $\Theta(\log n)$ bits suffice.
- 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 node u will flood exactly one message at each index i — in particular the smallest random number encountered by node u with message index i
- 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.
// 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:

- 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.
-

For all $u \in U$, $\mathcal{R}' \subseteq V_u \subseteq \mathcal{R}$. Notice that $\bar{s}_u(i)$ computed by u in line number 6 of Algorithm 1 is based on random numbers generated by all nodes in V_u . Therefore, at round t , node u is estimating \mathcal{R} using the exponential random numbers that were drawn by nodes in V_u . Since our adversary is oblivious, the choice of V_u is independent of the choice of the random numbers generated by each $v \in V_u$. Therefore, $\bar{s}_u(i)$ is an exponentially distributed random number with rate $|V_u| \geq \mathcal{R}'$ (cf. Property 2). For any $\delta > 2\beta$, let $\varsigma \leq \min\{\frac{\delta-2\beta}{1-\delta}, \frac{\delta}{1+\delta}\}$. When $P = \frac{3\gamma \ln n}{\varsigma^2} \in \Theta(\log n)$ parallel iterations are performed, where $\gamma \geq 1$, the required accuracy is obtained with probability $1 - \frac{1}{\Omega(n^\gamma)}$ (cf. Property 3). \square \square

3.5 Support Estimation Under an Adaptive Adversary

The algorithm for support estimation under an oblivious adversary (cf. Section 3.4) does not work under an adaptive adversary. To estimate the support of red nodes in the network, each red node draws a random number from the exponential distribution and floods it in an attempt to spread the smallest random number. When the adversary is adaptive, the smallest random numbers can easily be targeted and suppressed. To mitigate this difficulty, we consider a different algorithm in which the number of bits communicated is larger. In particular, the number of bits communicated per round by each node executing this algorithm is at most polynomial in n .

Let \mathcal{R} be the support of the red nodes. Every node floods its unique identifier along with a bit that indicates whether it is a red node or not. At most $\beta\sqrt{n}$ nodes' identifiers can be suppressed by the adversary for $\Omega(\log^2 n)$ rounds leaving at least $n - \beta\sqrt{n}$ unsuppressed identifiers (cf. Corollary 3.3). Each node counts the number of unique red identifiers A and non-red identifiers B that flood over it and estimates \mathcal{R} to be $A + \frac{n-A-B}{2}$.

This support estimation technique generalizes quite easily to arbitrary churn order. Therefore,

we state the following theorem more generally.

Theorem 3.8. *Consider the algorithm mentioned above in which nodes flood their unique identifiers indicating whether they are red nodes or not and assume that the network is controlled by an adaptive adversary. Let $c(n)$ be the order of the churn; we assume for simplicity that $c(n)$ is either n or \sqrt{n} . Then the following holds:*

1. *At least $n - \beta c(n)$ nodes estimate \mathcal{R} between $\mathcal{R} - \frac{\beta c(n)}{2}$ and $\mathcal{R} + \frac{\beta c(n)}{2}$. Furthermore, these nodes are aware that their estimate is within $\mathcal{R} - \frac{\beta c(n)}{2}$ and $\mathcal{R} + \frac{\beta c(n)}{2}$.*
2. *The remaining nodes are aware that their estimate of \mathcal{R} might fall outside $[\mathcal{R} - \frac{\beta c(n)}{2}, \mathcal{R} + \frac{\beta c(n)}{2}]$. When $c(n) = n$, it requires only $O(\log n)$ rounds, but when $c(n) = \sqrt{n}$, it requires $O(\log^2 n)$ rounds.*

Proof. Let u be any one of the $n - \beta c(n)$ nodes that receive at least $n - \beta c(n)$ unsuppressed identifiers (cf. Lemma 3.5 and Lemma 3.6). Let A and B be the number of unique identifiers from red nodes and non-red nodes, respectively, that flood over u . Let $C = n - A - B \leq \beta c(n)$. This means that u estimates \mathcal{R} to be $A + \frac{C}{2}$. Note that $A \leq \mathcal{R} \leq A + C$ and since $C \leq \beta c(n)$, \mathcal{R} is estimated between $\mathcal{R} - \frac{\beta c(n)}{2}$ and $\mathcal{R} + \frac{\beta c(n)}{2}$. Furthermore, since u received $n - \beta c(n)$ identifiers, it can be sure that its estimate is between $\mathcal{R} - \frac{\beta c(n)}{2}$ and $\mathcal{R} + \frac{\beta c(n)}{2}$.

If a node does not receive at least $n - \beta c(n)$ identifiers, then it is aware that its estimate of \mathcal{R} might not be within $[\mathcal{R} - \frac{\beta c(n)}{2}, \mathcal{R} + \frac{\beta c(n)}{2}]$.

From Lemma 3.5, when $c(n) = n$, the algorithm takes $O(\log n)$ rounds to complete because we want to ensure that unsuppressed nodes have flooded the network. When $c(n) = \sqrt{n}$, as a consequence of Lemma 3.6, the algorithm requires $O(\log^2 n)$ rounds. \square \square

4 STABLE AGREEMENT Under an Oblivious Adversary

In this section we will first present Algorithm 2 for the simpler problem of reaching BINARY CONSENSUS, where the input values are restricted to $\{0, 1\}$ (cf. Section 2.1). We will then use this algorithm as a subroutine for solving STABLE AGREEMENT in Section 4.2.

Throughout this section we assume suitable choices of ε and α such that the upper bound

$$(7) \quad \beta < \frac{1}{12}$$

can be satisfied for β ; note that (7) must hold in addition to bound (1). Moreover, we assume that a node can send and process up to $O(\log^2 m)$ bits in every round, where m is the size of the input value domain.

4.1 BINARY CONSENSUS

A node u that executes Algorithm 2 proceeds in a sequence of $O(\log n)$ checkpoints that are interleaved by $O(\log n)$ rounds. Each node u has a bit variable b_u that stores its current output value. At each checkpoint t_i , node u initiates support estimation of the number of nodes currently having 1 as their output bit by using the algorithm described in Section 3.4. (At checkpoint t_{R-1} , nodes estimate both: the support of 1 and 0.) The outcome of this support estimation will be available in checkpoint t_{i+1} where u has derived the estimation $\#(1)$. If u believes that the support of 1 is small ($\leq \frac{1}{4}n$), it sets its own output b_u to 0; if, on the other hand, $\#(1)$ is large ($\geq \frac{3}{4}n$),

u sets its output b_u to 1. This guarantees stability once agreement has been reached by a large number of nodes. When the support of 1 is roughly the same as the support of 0, we need a way to sway the decision to one side or the other. This is done by flooding the network whereby the flooding message of node v is weighted by some randomly chosen value, say r_v . The adversary can only guess which node has the highest weight and therefore, with constant probability, the flooding message with this highest weight (i.e., smallest random number) will be used to set the output bit by almost all nodes in the network.

Algorithm 2 BINARY CONSENSUS under an oblivious adversary; code executed by node u .

Let $decision_u$ be initialized to \perp .

Let b_u be the current output bit of u . If $u \in V^0$, then b_u is initialized to the input value of u ; otherwise it is set to \perp .

Let $t_1 = 1$ be the first checkpoint round. Subsequent checkpoint rounds are given by $t_i = t_{i-1} + O(\log n)$, for $i > 1$. For the terminating checkpoint t_R , we choose an $R \in O(\log n)$, i.e., $t_R \in O(\log^2 n)$.

At every checkpoint round t_i excluding t_R :

- 1: Initiate support estimation (to be completed in checkpoint round t_{i+1}).
- 2: Generate a random number r_u uniformly from $\{1, \dots, n^k\}$ for suitably large but constant k . (With high probability, we want exactly one node to have generated $\min_u r_u$.)
- 3: Initiate flooding of $\{r_u, b_u\}$ with terminating condition: $\langle (\text{HAS ENCOUNTERED ANOTHER MESSAGE INITIATED BY } v \neq u \text{ WITH } r_v < r_u) \vee (\text{CURRENT ROUND} \geq t_{i+1}) \rangle$.

At every checkpoint round t_i except t_1 :

- 4: Use the support estimation initiated at checkpoint round t_{i-1} . Let $\#(1)$ be u 's estimated support value for the number of nodes that had an output of 1.
- 5: **if** $\#(1) \leq \frac{1}{4}n$ **then**
- 6: $b_u \leftarrow 0$.
- 7: **else if** $\#(1) \geq \frac{3}{4}n$ **then**
- 8: $b_u \leftarrow 1$.
- 9: **else if** u has received flooded messages initiated in t_{i-1} **then**
- 10: Let $\{r_v, b_v\}$ be the message with the smallest random number that flooded over u .
- 11: $b_u \leftarrow b_v$.

At terminating checkpoint round t_R :

- 12: **if** $\#(1) \geq \frac{n}{2}$ **then**
- 13: $decision_u \leftarrow 1$.
- 14: Flood a 1-decision message ad infinitum.
- 15: **else if** $\#(0) \geq \frac{n}{2}$ **then**
- 16: $decision_u \leftarrow 0$.
- 17: Flood 0-decision message ad infinitum.

If u receives a b -decision message:

- 18: $decision_u \leftarrow b$
-

Theorem 4.1. Assume that the adversary is oblivious and that the churn limit per round is ϵn . Algorithm 2 reaches stability in $O(\log^2 n)$ rounds and achieves BINARY CONSENSUS with high probability.

Proof. Throughout this proof we repeatedly invoke the properties of the support estimation as stated in Theorem 3.7, which succeeds with probability $1 - 1/n^\gamma$. Assuming that $\gamma \geq 2$, suffices to guarantee that all of the $\Theta(\log n)$ invocations of the support estimation are accurate with high probability.

We first argue that Validity holds: Suppose that all nodes start with input value 1. The only way a node can set its output to 0 is by passing Line 5. This can happen for at most βn nodes. The only way that more nodes can set their output to 0 is if they estimate the support of 1 to be in $(\frac{1}{4}n, \frac{3}{4}n)$. If β is suitably small, Theorem 3.7 guarantees that with high probability this will not happen at any node. The argument is analogous for the case where all nodes start with 0.

Next we show Almost Everywhere Agreement: Let N_i be the number of nodes at checkpoint round t_i that output 1. Let LOW_i , HIGH_i , and MID_i , respectively, be the sets of nodes in V^{t_i} for which $\#(1) \leq \frac{1}{4}n$, $\#(1) \geq \frac{3}{4}n$, and $\frac{1}{4}n < \#(1) < \frac{3}{4}n$; note that nodes are placed in LOW_i , HIGH_i , and MID_i based on their $\#(1)$ values, which are estimates of N_{i-1} , not N_i . Clearly, we have that $\text{LOW}_i + \text{MID}_i + \text{HIGH}_i = n$.

For some $i > 1$, let $u^* \in V^{t_{i-1}}$ be the node that generated the smallest random number in checkpoint round t_{i-1} among all nodes in $V^{t_{i-1}}$. With high probability, u^* will be unique. By Corollary 3.4, with probability $1 - \beta$ (a constant), u^* is unsuppressed, implying that b_{u^*} will be used by all nodes in MID_i . Consider the following cases:

Case A ($N_{i-1} \leq (\frac{1}{4} - \delta)n$): From Theorem 3.7, we know that with high probability $|\text{LOW}_i| \geq (1 - \beta)n$ implying $|\text{MID}_i| + |\text{HIGH}_i| \leq \beta n$. Therefore, N_i will continue to be very small leading to small estimates $\#(1)$ in subsequent checkpoints. After $O(\log n)$ checkpoints, this causes at least $(1 - \beta)n$ nodes to decide on 0, with high probability. Moreover, it is easy to see that the remaining βn nodes will not be able to pass Line 12, since the adversary cannot artificially increase the estimated support of nodes with 1. (Recall from Section 3.4 that by suppressing the minimum random variables, the adversary can only make the estimate smaller.)

(We are presenting separate Cases B, C, and D for clarity. Equivalently, we could have treated them together as one case with the condition that $(\frac{1}{4} - \delta)n < N_{i-1} < (\frac{3}{4} + \delta)n$ leading to the implication that with high probability either $|\text{LOW}_i| + |\text{MID}_i| \geq (1 - \beta)n$ or $|\text{HIGH}_i| + |\text{MID}_i| \geq (1 - \beta)n$.)

Case B ($(\frac{1}{4} - \delta)n < N_{i-1} < (\frac{1}{4} + \delta)n$): With high probability, $|\text{LOW}_i| + |\text{MID}_i| \geq (1 - \beta)n$ implying $|\text{HIGH}_i| \leq \beta n$. Note first that nodes in LOW_i will set their output bits to 0. Since $N_{i-1} < (\frac{1}{4} + \delta)n$, there are at least $(\frac{3}{4} - \delta)n$ nodes in V^{t-1} that output 0. Of these, at most βn could have been suppressed. So, with probability at least $\frac{3}{4} - \delta - \beta$, u^* is an unsuppressed node that outputs 0. When u^* outputs 0, nodes in MID_i will set their output bits to 0. Thus, considering LOW_i and MID_i , we have at least $(1 - \beta)n$ nodes that set their output bits to 0 with constant probability. For a suitably small δ and $\beta < \frac{1}{4} - \delta$, this will lead to Case A in the next iteration, which means that subsequently nodes agree on 0.

Case C ($(\frac{1}{4} + \delta)n \leq N_{i-1} \leq (\frac{3}{4} - \delta)n$): With high probability, $|\text{MID}_i| \geq (1 - \beta)n$. With constant probability $(1 - \beta)$, u^* will be an unsuppressed node and nodes in MID_i will set their output bits to the same value b_{u^*} . This will lead to Case A in the next iteration.

Case D ($(\frac{3}{4} - \delta)n < N_{i-1} < (\frac{3}{4} + \delta)n$): This is similar to Case B, i.e., with constant probability, at least $(1 - \beta)n$ nodes will reach agreement on 1.

Case E ($N_{i-1} \geq (\frac{3}{4} + \delta)n$): This is similar to Case A. With high probability, at least $(1 - \beta)n$ nodes will decide on 1.

Note that, when a checkpoint falls either under Case A or Case E, with high probability, it will remain in that case. When a checkpoint falls under Case B, Case C, or Case D, with constant

probability, we get either Case A or Case E in the following checkpoint. Therefore, in $O(\log n)$ rounds, at least $(1 - \beta)n$ nodes will reach agreement with high probability and all other nodes will remain undecided.

For property Stability, note that if a node has decided on some value in checkpoint t_R , it continues to flood its decision message. We showed that, with high probability, at least $(1 - \beta)n$ nodes will decide on the same bit value. Therefore, it follows by Lemma 3.1 that agreement will be maintained ad infinitum among at least $(1 - \beta)n$ nodes. \square \square

4.2 A 3-phase Algorithm for STABLE AGREEMENT

We will now describe how we use Algorithm 2 as a building block for solving STABLE AGREEMENT: In order to use Algorithm 2 to solve STABLE AGREEMENT, we will need to make a couple of crucial adaptations.

- Suppose every vertex in V^0 has some auxiliary information. We can easily adapt Algorithm 2 so that when a node u decides on a bit value b , then, it also inherits the auxiliary information of some $v \in V^0$ whose initial bit value was b . This is guaranteed because our algorithm ensures Validity. The auxiliary information can be piggybacked on the messages that v generates throughout the course of the algorithm.
- For a typical agreement algorithm, we assume that all nodes simultaneously start running the algorithm. We want to adapt our algorithm so that only nodes in V^0 that have an initial output bit of 1 initiate the algorithm, while nodes that start with 0 are considered passive, i.e., these nodes do not generate messages themselves, but still forward flooding messages and start generating messages from the next checkpoint onward as soon as they notice that an instance of the algorithm is running.

We now sketch how the algorithm can be adapted: In the first checkpoint t_1 , each node v with a 1 initiates support estimation and flooding of message $\langle r_v, b_v = 1 \rangle$. If the number of nodes with 1 is small at checkpoint t_1 , then, at checkpoint t_2 , nodes that receive estimate values will conclude 0, which will get reinforced in subsequent checkpoints. However, if the number of nodes with a 1 at checkpoint t_1 is large (in particular, larger than βn), then, by suitable flooding, at least $(1 - \beta)n$ nodes will know that a support estimation is underway and will participate from checkpoint t_2 onward.

Selection and Flooding Phase: In the very first round, each node $u \in V^0$ generates a uniform random number r_u from $(0, 1)$ and, if the random number is less than $\frac{4 \log n}{n}$, u becomes a *candidate* and initiates a message m_u for flooding. The message m_u contains the random number r_u and the general value VAL_u (from domain $\{0, \dots, m\}$) assigned to u by the adversary. This phase ends after $\Theta(\log n)$ rounds to ensure that no more than βn nodes are suppressed (the precise bound on the number of rounds is given by Corollary 3.3). The flooding of the generated messages, however, goes on ad infinitum.

Candidate Elimination Phase: We initiate $\Theta(\log n)$ parallel iterations of BINARY CONSENSUS, whereby each iteration is associated with one of the $\Theta(\log n)$ flooding messages, generated by the candidates in the first phase. More precisely, the i -th instance of BINARY CONSENSUS for the i -th candidate and its flooding message m_{u_i} is designed as follows: nodes that have received a flooded

message m_{u_i} , set their input bit (of the i -th instance of BINARY CONSENSUS) to 1 and initiate BINARY CONSENSUS. We say that a flooded message m_u is a *survived candidate* message if the instance of BINARY CONSENSUS associated with it reached a decision value of 1.

Confirmation Phase: Among the survived candidate messages, every node v chooses the message m_{u_j} among its received messages that has the smallest random number r_{u_j} (and associated general input value val_{u_j}), and initiates a support estimation for the number of nodes that have received m_{u_j} . If the support estimation reveals a support of at least $(1 - \beta - \delta)n$ for m_j then v decides on val_{u_j} . Nodes keep flooding their decision ad infinitum.

Theorem 4.2. *Consider the oblivious adversary and suppose that εn nodes can be subject to churn in every round. The 3-phase algorithm is correct with high probability and reaches STABLE AGREEMENT in $O(\log^2 n)$ rounds.*

Proof. Validity follows immediately from the fact that nodes only decide on some value that was the input value of a (survived) candidate.

We now argue Almost Everywhere Agreement: Since all nodes choose independently whether to become candidate, a simple application of a standard Chernoff bound shows that the number of candidates is in the range $[2 \log n, 8 \log n]$ with probability $\geq 1 - n^{-3}$; in the remainder of this proof, we condition on this event to be true.

Consider the message m_u generated by some candidate u in the Selection and Flooding phase, and consider its associated instance of BINARY CONSENSUS: If m_u has reached at least $(1 - \beta)n$ nodes by flooding, it follows by the properties of the BINARY CONSENSUS algorithm that the decision value of BINARY CONSENSUS will be 1 with probability $1 - n^{-2}$. On the other hand, if m_u has a very small support (say, βn), the consensus value will be 0 with probability $1 - n^{-3}$ (cf. Case A of the proof of Theorem 4.1), and, if the support of m_u is neither too small nor too large, the nodes will reach consensus on either 0 or 1. Thus we can interpret a decision of 1 regarding the i -th message, as a confirmation that the i -th candidate had sufficiently large support. By taking a union bound, it follows that, with probability at least $1 - n^{-2}$, at least $(1 - \beta)n$ nodes agree on the set of survived candidate messages, since they reached agreement in each iteration of BINARY CONSENSUS. Since the adversary is oblivious, each of the $\Theta(\log n)$ flooding messages generated by the candidates will not be suppressed with probability at least $(1 - \beta)$ (cf. Corollary 3.4). Therefore, with probability $\geq 1 - n^{-2}$, at least one candidate u will have $|\text{INFL}(u, O(\log n))| \geq (1 - \beta)n$ and thus the set of survived candidates S will be nonempty; let $w \in S$ be the candidate who generated the smallest random number. When the support estimation is initiated in the third phase, a set of at least $(1 - \beta)n$ nodes will measure w 's support to be at least $(1 - \beta - \delta)n$ for some $\delta > 2\beta$ with probability $\geq 1 - n^{-2}$ (cf. Theorem 3.7) and decide on the value val_w of w , whereas nodes that do not observe high support remain undecided. This shows Almost Everywhere Agreement.

Analogously to Algorithm 2, nodes in S flood their decision messages, which are adopted by newly incoming nodes. By virtue of Lemma 3.1, the stability property is maintained ad infinitum.

The additional running time overhead of the above three phases excluding Algorithm 2 is only in $O(\log n)$. This completes the proof of the Theorem. \square \square

5 STABLE AGREEMENT Under an Adaptive Adversary

In this section we consider the STABLE AGREEMENT problem while dealing with a more powerful adaptive adversary. At the beginning of a round r , this adversary observes the entire state of the network and previous communication between nodes (including even previous outcomes of random choices!), and thus can adapt its choice of G^r to make it much more difficult for nodes to achieve agreement.

It is instructive to consider the algorithms presented in Section 4 in this context. Both approaches are doomed to fail in the presence of an adaptive adversary: For the STABLE AGREEMENT algorithm, the expected number of nodes that initiate flooding in the flooding phase is $\log n$. Even though each of these nodes would have expanded its influence set to some constant size by the end of the next round, the adaptive adversary can spot and immediately churn out all these nodes before they can communicate with anyone else, thus none of these values will gain any support.

Algorithm 2 fails for the simple reason that the adversary can selectively suppress the flooding of the smallest generated random value $z \in \{1, \dots, n^k\}$ with attached bit b_z from ever reaching some 50% of the nodes, which instead might use a distinct minimum value z' (with an attached bit value $b_{z'} \neq b_z$) to guide their output changes.

To counter the difficulties we have mentioned, we relax the model. Firstly, we limit the order of the churn to \sqrt{n} . Secondly, we allow messages of up to a polynomial (in n) number of bits to be sent over a link in a single round. Under these relaxations, we can estimate the support of red nodes in the network simply by flooding all the unique identifiers of the red and non-red nodes (cf. Theorem 3.8).

Similarly to Section 4, we will first solve BINARY CONSENSUS under these assumptions and then show how to implement STABLE AGREEMENT. In this section we assume that the number of nodes in the network is sufficiently large, such that

$$(8) \quad n \gg 4\beta^2.$$

Moreover, we assume that every node can send and process up to $O(n^c + \log m)$ bits per round, where c is a constant and m is the size of the input domain.

5.1 BINARY CONSENSUS

We now describe an algorithm for solving BINARY CONSENSUS, which is similar in spirit to Algorithm 2. The main difference is the handling of the case where the support of the nodes that output 1 is roughly equal to the support of the nodes with output bit 0. In this case we rely on the variance of random choices made by individual nodes to sway the balance of the support towards one of the two sides with constant probability.

First, we argue why this technique does not work when the churn limit is $\omega(\sqrt{n})$: In our algorithm we handle the case where the support of 0 and 1 is roughly equal, by causing each node to update its current output bit to the outcome of a (private) unbiased coin flip. The standard deviation that we get for the sum of these individual random variables is $O(\sqrt{n})$ and the event where the balance is swayed by $O(\sqrt{n})$ occurs with constant probability. But since the adversary is adaptive and has $\omega(\sqrt{n})$ churn to play with, it can immediately undo this favourable imbalance by churning out nodes such that the support of 0 and 1 will yet again be roughly equal.

Theorem 5.1. *Algorithm 3 solves BINARY CONSENSUS with high probability and reaches stability within $O(\log^3 n)$ rounds, in the presence of an adaptive adversary and up to $\varepsilon\sqrt{n}$ churn per round.*

Algorithm 3 BINARY CONSENSUS under an adaptive adversary; code executed by node u .

Let $decision_u$ be initialized to \perp .

Let b_u be the current output bit of u . If $u \in V^0$, then b_u is initialized to the input value of u ; otherwise it is set to \perp .

Let $t_1 = 1$ be the first checkpoint round. Subsequent checkpoint rounds are given by $t_i = t_{i-1} + O(\log^2 n)$, for $i > 1$, with time between consecutive checkpoint rounds sufficient for unsuppressed nodes to reach a common influence (cf. Lemma 3.6). For the terminating checkpoint t_R , we choose an $R \in O(\log n)$, i.e., $t_R \in O(\log^3 n)$.

At every checkpoint round t_i excluding t_R :

- 1: Initiate support estimation (to be completed in checkpoint round t_{i+1}).

At every checkpoint round t_i excluding t_1, t_R :

- 2: Use the support estimation initiated at checkpoint round t_{i-1} . Let $\#(1)$ be the estimated support value for nodes that output 1.
- 3: **if** support estimation is not accurate within $[\mathcal{R} - \frac{\beta\sqrt{n}}{2}, \mathcal{R} + \frac{\beta\sqrt{n}}{2}]$ **then**
- 4: Do nothing.
- 5: **else if** $\#(1) < \frac{n}{2} - \frac{\beta\sqrt{n}}{2}$ **then**
- 6: $b_u \leftarrow 0$.
- 7: **else if** $\#(1) > \frac{n}{2} + \frac{\beta\sqrt{n}}{2}$ **then**
- 8: $b_u \leftarrow 1$.
- 9: **else**
- 10: **if** the outcome of an unbiased coin flip is heads **then**
- 11: $b_u \leftarrow 0$.
- 12: **else**
- 13: $b_u \leftarrow 1$.

At terminating checkpoint round t_R :

- 14: **if** $\#(1) \geq \frac{n}{2}$ **then**
- 15: $decision_u \leftarrow 1$.
- 16: Flood a 1-decision message ad infinitum.
- 17: **else if** $\#(0) \geq \frac{n}{2}$ **then**
- 18: $decision_u \leftarrow 0$.
- 19: Flood a 0-decision message ad infinitum.

If u receives a b -decision message:

- 20: $decision_u \leftarrow b$
-

Proof. First consider the Validity property: Suppose that all nodes start with input value 1. Theorem 3.8 guarantees that any node u that receives insufficiently many identifiers for support estimation, will execute Line 4 and therefore never set its output to 0. On the other hand, if u does receive sufficiently many samples, again Theorem 3.8 ensures that it will always pass the if-check in Line 7. Thus, no node can ever output 0. The case where all nodes start with 0 can be argued analogously.

Next, we will show that Algorithm 3 satisfies Almost Everywhere Agreement. Let N_i be the number of vertices at checkpoint round t_i with output bit 1. Let LOW_i , $HIGH_i$, and MID_i , respectively, be the sets of nodes in V^{t_i} for which $\#(1) \leq n/2 - \frac{\beta\sqrt{n}}{2}$, $\#(1) \geq n/2 + \frac{\beta\sqrt{n}}{2}$, and $n/2 - \frac{\beta\sqrt{n}}{2} < \#(1) < n/2 + \frac{\beta\sqrt{n}}{2}$; note that nodes are placed in LOW_i , $HIGH_i$, and MID_i based on their $\#(1)$ values, which are estimates of N_{i-1} , not N_i . In a slight abuse of notation, we use LOW_i , MID_i , and $HIGH_i$ to also refer to their respective cardinalities. Clearly, we have that

$$LOW_i + MID_i + HIGH_i = n.$$

Furthermore, observe that either LOW_i or HIGH_i will be 0. Otherwise, we will have two nodes such that one estimates N_{i-1} below $n/2 - \frac{\beta\sqrt{n}}{2}$, while the other estimates it above $n/2 + \frac{\beta\sqrt{n}}{2}$ — a violation of Theorem 3.8.

Consider the following cases:

Case A ($N_{i-1} < n/2 - \beta\sqrt{n}$): From Theorem 3.8, $\text{LOW}_i \geq n - \beta\sqrt{n}$ and all nodes in LOW_i will set themselves to output 0. Once this case is reached in some checkpoint, it will be reached in all future checkpoints until t_R with high probability. Therefore, the algorithm guarantees Almost Everywhere Agreement on 0 in t_R ; with high probability, nodes do not pass Line 14 in checkpoint t_R , thus no node will ever decide on 1.

Case B ($N_{i-1} > n/2 + \beta\sqrt{n}$): This case is similar to Case A with the difference that almost all nodes decide on 1.

Case C ($n/2 - \beta\sqrt{n} \leq N_{i-1} \leq n/2$): Notice that $\text{HIGH}_i = 0$. Therefore,

$$(9) \quad \text{LOW}_i + \text{MID}_i \geq n - \beta\sqrt{n}.$$

We consider two subcases:

1. In this case, we assume that LOW_i is at least $n/2 + \beta\sqrt{n}$. This will set $N_i < n/2 - \beta\sqrt{n}$ putting the network in Case A in the next checkpoint.

2. In this case, we assume that $\text{LOW}_i < n/2 + \beta\sqrt{n}$. This implies that

$$\text{MID}_i \geq n - \text{LOW}_i - \beta\sqrt{n} \geq n/2 - 2\beta\sqrt{n}.$$

The nodes in MID_i will choose 1 or 0 with equal probability. The number of nodes that choose 0 is a binomial distribution with mean $\frac{\text{MID}_i}{2}$ and standard deviation $\frac{\sqrt{\text{MID}_i}}{2}$. Clearly, with some constant probability, $\frac{\text{MID}_i}{2} + \frac{\sqrt{\text{MID}_i}}{2}$ or more nodes in the set MID_i will set themselves to output 0. Therefore, with constant probability,

$$\begin{aligned} N_i &< n - \text{LOW}_i - \frac{\text{MID}_i}{2} - \frac{\sqrt{\text{MID}_i}}{2} \\ &< n - \text{LOW}_i - \frac{n - \text{LOW}_i - \beta\sqrt{n}}{2} \\ &\quad - \frac{\sqrt{n - \text{LOW}_i - \beta\sqrt{n}}}{2} \end{aligned}$$

Clearly, $N_i < \frac{n}{2} - \beta\sqrt{n}$ if

$$3\beta\sqrt{n} < \sqrt{n - \text{LOW}_i - \beta\sqrt{n}},$$

which means that

$$\text{LOW}_i + \beta\sqrt{n} < n - 9\beta^2 n.$$

We know that $\text{LOW}_i < \frac{n}{2} + \beta\sqrt{n}$. Therefore, $N_i < \frac{n}{2} - \beta\sqrt{n}$ if

$$\frac{n}{2} + 2\beta\sqrt{n} < n - 9\beta^2n,$$

that is,

$$2\beta < \sqrt{n} \left(\frac{1}{2} - 9\beta^2 \right).$$

In other words, as long as

$$(10) \quad n > \frac{4\beta^2}{\left(\frac{1}{2} - 9\beta^2 \right)^2},$$

it holds with constant probability that

$$N_i < \frac{n}{2} - \beta\sqrt{n},$$

which will put the network in Case A at the next checkpoint round. Assumption (8) guarantees that Condition (10) is easily met.

Case D ($n/2 < N_{i-1} \leq n/2 + \beta\sqrt{n}$): Using arguments similar to Case C, we can show that with constant probability,

$$N_i > \frac{n}{2} + \beta\sqrt{n},$$

thereby, putting the network in Case B.

Clearly, after $O(\log n)$ checkpoint rounds the network will reach either Case A or Case B⁶ with high probability and hence achieve Almost Everywhere Agreement on either 0 or 1.

For property Stability, note that if a node has decided on some value $\neq \perp$ in checkpoint t_R , it continues to flood its decision message. Since at least $(1-\beta)n$ have decided, it follows by Lemma 3.1 that any nodes that have been churned in will also decide on this value within a constant number of rounds, thus agreement will be maintained ad infinitum. \square \square

5.2 STABLE AGREEMENT

Now that we have a solution for BINARY CONSENSUS, we will show how to use it to solve STABLE AGREEMENT where nodes have input values from some set $\{0, \dots, m\}$, for $m \geq 1$. Given some input value VAL we can write it in the base-2 number system as $(b_0, \dots, b_{\log m})$ where $b_i \in \{0, 1\}$, for $0 \leq i \leq \log m$. We call VAL a *general input value* and b_i a *binary input value*.

The basic idea of the STABLE AGREEMENT algorithm is to run an instance of the BINARY CONSENSUS algorithm for each b_i and then combine the agreed bits $d_1, \dots, d_{\log m}$ to obtain agreement on the general input values. We now describe our algorithm; the detailed pseudo code is presented in Algorithm 4. Consider the i -th iteration of Algorithm 4 and suppose that d_1, \dots, d_{i-1} are the first $i-1$ decision values of the previous $i-1$ iterations of the BINARY CONSENSUS algorithm. We say that a node u *knows a general input value matching the first i binary decision values*, if u

⁶Due to Equation (8) we know that Cases A and B exist.

has knowledge of a some $\text{VAL} \in \{0, \dots, m\}$ that was the input value of some node v and the first $i - 1$ bits of VAL are exactly d_1, \dots, d_{i-1} . We denote the i -th bit value of a general value VAL by $\text{VAL}[i]$. Recall that the BINARY CONSENSUS algorithm executes the support estimation routines developed in Section 3.5. We slightly modify the support estimation routine by requiring each node u to also piggyback its current general value VAL_u onto the message it generates for support estimation. Moreover, when u floods the decision message of the BINARY CONSENSUS algorithm, it also piggybacks VAL_u . Whenever a node v updates its current output bit value to b , this guarantees that v has learned of a general value VAL_w that has b as its first bit. Thus v sets VAL_v to the new value VAL_w and chooses its next input value for the $(i + 1)$ -th iteration of the BINARY CONSENSUS algorithm to be the $(i + 1)$ -th bit of VAL_v . This is formalized in the following lemma:

Lemma 5.2. *Consider iteration i of the BINARY CONSENSUS subroutine executed in Algorithm 4. If a node u has a current binary output value of b , then the i -th bit of VAL_u is b .*

Proof. We will show the result by induction over the iterations of the BINARY CONSENSUS algorithm. Initially, in the first iteration, node u uses the first bit of its input value VAL_u . Now suppose that u sets its output bit to $1 - b$ at some point during the first iteration. We say that u violates *general validity*. There are two possible cases: In the first case, u observed a sufficiently large support for $1 - b$ and thus received a support estimation message generated by a node v that had a current output bit $1 - b$, while in the second case u received a decision message generated by v . In either case, it follows from the description of the algorithm that node v has piggybacked VAL_v on top of this message. If $\text{VAL}_v[i] = 1 - b$, then v has updated its own output bit without updating VAL_v , due to receiving some message from another node v' , and both nodes, v and v' , violate general validity. By backwards traversing this chain of nodes that violate general validity, we eventually reach a node w which has set its output bit value to $1 - b$ but $\text{VAL}_w[i] = b$, without having received a message from a node that violates general validity. According to the BINARY CONSENSUS algorithm, w only sets its bit value to $1 - b$ if it has either observed sufficient support for $1 - b$ or received a decision message containing a value of $1 - b$. In both cases, it follows from the description of the algorithm that w updates VAL_w to the piggybacked general value, the i -th bit of which is $1 - b$, providing a contradiction. \square \square

The above lemma guarantees that we can combine the decision bits of the BINARY CONSENSUS iterations to get a general decision value that satisfies validity. We can therefore show the following theorem:

Theorem 5.3. *Suppose that the network is controlled by an adaptive adversary who can subject up to $\varepsilon\sqrt{n}$ nodes to churn in every round. There is an algorithm that solves STABLE AGREEMENT with high probability and reaches stability in $O(\log m \log^3 n)$.*

Proof. Almost-everywhere agreement follows almost immediately from the fact that the BINARY CONSENSUS algorithm satisfies almost-everywhere agreement; what remains to be shown is that all except $\beta\sqrt{n}$ nodes decide: Note that it is possible that a set S of $\beta\sqrt{n}$ nodes can remain undecided when running an instance of the BINARY CONSENSUS algorithm. The nodes in S will not propose any values in the next iteration but will participate in the support estimation and the propagation of messages. By the correctness of the BINARY CONSENSUS algorithm, all except $\beta\sqrt{n}$ nodes eventually know the decision bit d_i of the i -th iteration. In the next iteration, any node v that knows the decision bit d_{i+1} , also knows a general value VAL such that $\text{VAL}[i + 1] = d_{i+1}$ and

Algorithm 4 Solving STABLE AGREEMENT using BINARY CONSENSUS. Pseudo code for node u .

- 1: Suppose that node u starts with an initial general input value VAL_u .
 - 2: **for** $i \leftarrow 0$ to $\log m$ **do**
 - 3: Node u initiates BINARY CONSENSUS by proposing the i -th bit of its current VAL_u . Recall that BINARY CONSENSUS will be reached in $O(\log^3 n)$ rounds.
 - 4: When participating in the support estimation that is part of BINARY CONSENSUS each node u piggybacks its VAL_u on top of the support estimation message.
 - 5: Let d_i be the decision returned by BINARY CONSENSUS algorithm. If node u has decided on bit value $d_i \in \{0, 1\}$, then u has learned of a general input value VAL where the i -th bit is d_i : Node u updates its current value VAL_u by setting it to VAL and floods VAL_u along the decision message according to the BINARY CONSENSUS algorithm.
 - 6: If u did not decide in the BINARY CONSENSUS algorithm, then u does not propose a value in the $(i + 1)$ -th iteration.
 - 7: If u did not decide in the last iteration, it remains undecided. Otherwise, u returns the VAL_u as its decision value and floods this value ad infinitum.
-

thus can propose in the subsequent iteration. This holds regardless of whether $v \in S$ and thus all except $\beta\sqrt{n}$ nodes participate in each iteration.

For validity, we argue that Algorithm 4 maintains the following invariant at the end of every iteration i : a node that is aware of the decision (bit) values of the first i runs of the BINARY CONSENSUS subroutine, has knowledge of a general value matching the first i binary decision values. By Lemma 5.2, it follows that if a node u proposes a bit b in iteration i , then b is the i -th bit of some general input value VAL . This guarantees that the sequence of decision bits correspond to some general input value and thus satisfies validity.

Finally, we observe that the proof of stability is identical to the BINARY CONSENSUS algorithm, thus completing the proof. □ □

6 Impossibility of a Deterministic Solution

In this section we show that there is no deterministic algorithm to solve STABLE AGREEMENT even when the churn is restricted to only a constant number of nodes per round. As a consequence, randomization is a necessity for solving STABLE AGREEMENT.

We introduce some well known standard notations (see [7, Chap. 5]) used for showing impossibility results of agreement problems. The *configuration* C^r of the network at round r consists of

- the graph of the network at that point in time, and
- the local state of each node in the network.

A specific run ρ of some STABLE AGREEMENT algorithm \mathcal{A} is entirely determined by an infinite sequence of configurations C^0, C^1, \dots where C^0 contains the initial state of the graph before the first round. Consider the input value domain $\{0, 1\}$. A configuration C^r is *1-valent* (resp., *0-valent*) if all possible runs of \mathcal{A} that share the common prefix up to and including C^r , lead to an agreement value of 1 (resp., 0). Note that this decision value refers to the decision of the large majority of nodes; strictly speaking, a small fraction of nodes might remain undecided on \perp . A configuration is *univalent* if it is either 1-valent or 0-valent. Any configuration that is not univalent is called a *bivalent* configuration. The following observation follows immediately from the definition of the STABLE AGREEMENT problem.

Observation 1. *Consider a bivalent configuration C^r in round r reached by an algorithm \mathcal{A} that solves STABLE AGREEMENT and ensures Almost Everywhere Agreement. No node in V^r can have decided on a value $\neq \perp$ by round r .*

Theorem 6.1. *Suppose that the sequence of graphs $(G^r)_{r \geq 0}$ is an expander family with maximum degree Δ . Assume that the churn is limited to at most $\Delta+1$ nodes per round. There is no deterministic algorithm that solves STABLE AGREEMENT if the network is controlled by an adaptive adversary.*

Proof. We use an argument that is similar to the argument used in the proof that $f+1$ rounds are required for consensus in the presence of f faults (cf. [7, Chap. 5]). For the purpose of this impossibility proof, we restrict the input domain of nodes to $\{0, 1\}$ and allow arbitrary congestion on the communication channels. Moreover, we assume that the topology of the network is fixed throughout the run. Thus the adversary can only “replace” nodes at the same position by some other nodes.

For the sake of contradiction, assume that such a deterministic algorithm \mathcal{A} exists that solves STABLE AGREEMENT under the assumed settings. We will prove our theorem by inductively constructing an infinite run ρ of this algorithm consisting of a sequence of bivalent configurations. By virtue of Observation 1 this allows us to conclude that nodes do not reach almost everywhere agreement.

To establish the basis of our induction, we need to show that there is an initial bivalent configuration C^0 at the start of round 1. Assume in contradiction that there is no bivalent starting configuration. Let D_0 (resp. D_1) be the configuration where all nodes start with a value 0 (resp., 1); note that by validity the decision value must be on 0 (resp., 1). Consider the sequence of configurations starting at D_0 and ending at D_1 where the only difference between any two configurations adjacent (in this sequence) is a single bit, i.e., exactly 1 node has a different input value. Since D_0 is 0-valent and D_1 is 1-valent, this implies that there are two possible starting configurations in this sequence, C_0^0 and C_1^0 , in which (i) the input values are the same for all but one node u^0 , but (ii) C_0^0 is 0-valent whereas C_1^0 is 1-valent. Consider the respective one-round extension of C_0^0 and C_1^0 where the adversary simply churns out node u^0 . Both successor configurations C_0^1 and C_1^1 are indistinguishable for all other nodes, in particular they have no way of knowing what initial value was assigned to u^0 , since all witnesses have been removed by the adversary. Therefore, C_0^1 and C_1^1 must both be either 0-valent or 1-valent, a contradiction. This shows that there is an initial bivalent configuration, thereby establishing the basis for our induction.

For the inductive step, we assume that the network is in a bivalent configuration C^{r-1} at the end of round $r-1$. We will extend C^{r-1} by one round (guided by the adversary) that yields another bivalent configuration C^r . Assume for the sake of a contradiction that every possible one-round extension of C^{r-1} yields a univalent configuration. Without loss of generality, assume that the one-round extension γ where no node is churned out is 1-valent and yields configuration C_1^r . Since by assumption C^{r-1} was bivalent, there is another one-round extension γ' that yields a 0-valent configuration C_0^r . Moreover, we know that a nonempty set S of size at most $\Delta+1$ nodes must have been subject to churn in γ' . (This is the only difference between C_0^r and C_1^r — recall that the edges of the graph are stable throughout the run.)

Let S' be a subset of S and let $\gamma_{S'}$ be the one-round extension of C^{r-1} that we get when only nodes in S' are churned out. Clearly, $\gamma = \gamma_\emptyset$ and $\gamma' = \gamma_S$. Consider the lattice of all such one-round extension bounded by γ and γ' that is given by the power set of S . Starting at γ and moving

towards γ' along some path, we must reach a one-round extension $\gamma_{\{v_1, \dots, v_k\}}$ that yields a 1-valent configuration D_1^r , whereas the next point on this path is some one-round extension $\gamma_{\{v_1, \dots, v_{k+1}\}}$ that ends in a 0-valent configuration D_0^r . The only difference between these two extensions is that node v_{k+1} is churned out in the latter but not in the former extension. Now consider the one-round extensions of D_0^r and D_1^r where v_{k+1} and all its neighbors are churned out, yielding D_0^{r+1} and D_1^{r+1} . For all other nodes, D_0^r and D_1^r are indistinguishable and therefore they must either both be 0-valent or both be 1-valent. This, however, is a contradiction. \square \square

Considering that expander graphs usually are assumed to have constant degree, Theorem 6.1 implies that even if we limit the churn to a constant, the adaptive adversary can still beat any deterministic algorithm.

7 Conclusion

We have introduced a novel framework for analyzing highly dynamic distributed systems with churn. We believe that our model captures the core characteristics of such systems: a large amount of churn per round and a constantly changing network topology. Future work involves extending our model to include Byzantine nodes and corrupted communication channels. Furthermore, our work raises some key questions: How much churn can we tolerate in an adaptive setting? Are there algorithms that tolerate linear (in n) churn in an adaptive setting? We show that we can tolerate $O(\sqrt{n})$ churn in an adaptive setting, but it takes a polynomial (in n) number of communication bits per round. An intriguing problem is to reduce the number of bits to polylogarithmic in n .

While the main focus of this paper was achieving agreement among nodes which is one of the most important tasks in a distributed system, as a next step, it might be worthwhile to investigate whether the techniques presented in this paper can serve as useful building blocks for tackling other important tasks like aggregation or leader election in highly dynamic networks.

References

- [1] Yehuda Afek, Baruch Awerbuch, and Eli Gafni. Applying static network protocols to dynamic networks. In *FOCS'87*, pages 358–370, 1987.
- [2] Yehuda Afek, Eli Gafni, and Adi Rosen. The slide mechanism with applications in dynamic networks. In *ACM PODC*, pages 35–46, 1992.
- [3] R. Ahlswede, N. Cai, S. Li, and R. Yeung. Network information flow. *Transactions on Information Theory*, 46(4):1204–1216, 2000.
- [4] James Aspnes, Navin Rustagi, and Jared Saia. Worm versus alert: Who wins in a battle for control of a large-scale network? In *OPODIS*, pages 443–456, 2007.
- [5] James Aspnes and Gauri Shah. Skip graphs. In *SODA*, pages 384–393, 2003.
- [6] James Aspnes and Udi Wieder. The expansion and mixing time of skip graphs with applications. In *SPAA*, pages 126–134, 2005.
- [7] Hagit Attiya and Jennifer Welch. *Distributed Computing: Fundamentals, Simulations and Advanced Topics (2nd edition)*. John Wiley Interscience, March 2004.

- [8] John Augustine, Gopal Pandurangan, and Peter Robinson. Fast byzantine agreement in dynamic networks. In *PODC*, pages 74–83, 2013.
- [9] Chen Avin, Michal Koucký, and Zvi Lotker. How to explore a fast-changing world (cover time of a simple random walk on evolving graphs). In *ICALP*, pages 121–132, 2008.
- [10] B. Awerbuch and F. T. Leighton. Improved approximation algorithms for the multi-commodity flow problem and local competitive routing in dynamic networks. In *ACM STOC*, pages 487–496, May 1994.
- [11] Baruch Awerbuch, Boaz Patt-Shamir, David Peleg, and Michael E. Saks. Adapting to asynchronous dynamic networks. In *STOC'92*, pages 557–570, 1992.
- [12] Baruch Awerbuch and Christian Scheideler. Group Spreading: A Protocol for Provably Secure Distributed Name Service. In *ICALP*, pages 183–195, 2004.
- [13] Baruch Awerbuch and Christian Scheideler. The hyperring: a low-congestion deterministic data structure for distributed environments. In *SODA*, pages 318–327, 2004.
- [14] Baruch Awerbuch and Christian Scheideler. Robust random number generation for peer-to-peer systems. In *OPODIS*, pages 275–289, 2006.
- [15] Baruch Awerbuch and Christian Scheideler. Towards a scalable and robust DHT. *Theory of Computing Systems*, 45:234–260, 2009.
- [16] 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.
- [17] Hervé Baumann, Pierluigi Crescenzi, and Pierre Fraigniaud. Parsimonious flooding in dynamic graphs. In *PODC*, pages 260–269, 2009.
- [18] Piotr Berman and Juan A. Garay. Fast consensus in networks of bounded degree. *Distributed Computing*, 7(2):67–73, 1993.
- [19] John F. Canny. Collaborative filtering with privacy. In *IEEE Symposium on Security and Privacy*, pages 45–57, 2002.
- [20] 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.
- [21] 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.
- [22] Keren Censor Hillel and Hadas Shachnai. Partial information spreading with application to distributed maximum coverage. In *Proceedings of the 29th ACM SIGACT-SIGOPS Symposium on Principles of Distributed Computing*, PODC '10, pages 161–170, New York, NY, USA, 2010. ACM.

- [23] Hyun Chul Chung, Peter Robinson, and Jennifer L. Welch. Optimal regional consecutive leader election in mobile ad-hoc networks. *FOMC '11*, pages 52–61. ACM, 2011.
- [24] Andrea Clementi, Riccardo Silvestri, and Luca Trevisan. Information spreading in dynamic graphs. In *PODC*, 2012.
- [25] Website of Cloudmark Inc. <http://cloudmark.com/>.
- [26] Edith Cohen. Size-estimation framework with applications to transitive closure and reachability. *J. Comput. Syst. Sci.*, 55(3):441–453, 1997.
- [27] A. Das Sarma, A. Molla, and G. Pandurangan. Fast distributed computation in dynamic networks via random walks. In *DISC*, 2012.
- [28] 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.
- [29] A. Dembo and O. Zeitouni. Large deviations techniques and applications. *Elearn*, 1998.
- [30] 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.
- [31] Shlomi Dolev. *Self-stabilization*. MIT Press, Cambridge, MA, USA, 2000.
- [32] Cynthia Dwork, David Peleg, Nicholas Pippenger, and Eli Upfal. Fault tolerance in networks of bounded degree. *SIAM J. Comput.*, 17(5):975–988, 1988.
- [33] 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.
- [34] Amos Fiat, Steve Gribble, Anna Karlin, Jared Saia, and Stefan Saroiu. Dynamically Fault-Tolerant Content Addressable Networks. In *Proceedings of the First International Workshop on Peer-to-Peer Systems*, Cambridge, MA, 2002.
- [35] Amos Fiat and Jared Saia. Censorship resistant peer-to-peer content addressable networks. In *SODA*, pages 94–103, 2002.
- [36] Amos Fiat, Jared Saia, and Maxwell Young. Making chord robust to byzantine attacks.
- [37] E. Gafni and B. Bertsekas. Distributed algorithms for generating loop-free routes in networks with frequently changing topology. *IEEE Trans. Comm.*, 29(1):11–18, 1981.
- [38] 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.
- [39] C.M. Grinstead and J.L. Snell. *Introduction to probability*. American Mathematical Society, 1997.

- [40] 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.
- [41] Bernhard Haeupler and David Karger. Faster information dissemination in dynamic networks via network coding. In *ACM PODC*, pages 381–390, 2011.
- [42] Kirsten Hildrum and John Kubiawicz. Asymptotically efficient approaches to fault-tolerance in peer-to-peer networks. In *DISC*, volume 2848 of *Lecture Notes in Computer Science*, pages 321–336. Springer, 2003.
- [43] Riko Jacob, Andréa W. Richa, Christian Scheideler, Stefan Schmid, and Hanjo Täubig. A distributed polylogarithmic time algorithm for self-stabilizing skip graphs. In *PODC*, pages 131–140, 2009.
- [44] 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.
- [45] 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.
- [46] Valerie King, Jared Saia, Vishal Sanwalani, and Erik Vee. Scalable leader election. In *SODA*, pages 990–999, 2006.
- [47] 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.
- [48] F. Kuhn and R. Oshman. Dynamic networks: Models and algorithms. *SIGACT News*, 42(1):82–96, 2011.
- [49] Fabian Kuhn, Nancy Lynch, and Rotem Oshman. Distributed computation in dynamic networks. In *ACM STOC*, pages 513–522, 2010.
- [50] Fabian Kuhn, Rotem Oshman, and Yoram Moses. Coordinated consensus in dynamic networks. *PODC '11*, pages 1–10. ACM, 2011.
- [51] Fabian Kuhn, Stefan Schmid, and Roger Wattenhofer. Towards worst-case churn resistant peer-to-peer systems. *Distributed Computing*, 22(4):249–267, 2010.
- [52] C. Law and K.-Y. Siu. Distributed construction of random expander networks. In *INFOCOM 2003*, volume 3, pages 2133 – 2143 vol.3, march-3 april 2003.
- [53] Nancy Lynch. *Distributed Algorithms*. Morgan Kaufman Publishers, Inc., San Francisco, USA, 1996.
- [54] Peter Mahlmann and Christian Schindelhauer. Peer-to-peer networks based on random transformations of connected regular undirected graphs. In *SPAA*, pages 155–164, 2005.
- [55] David J. Malan and Michael D. Smith. Host-based detection of worms through peer-to-peer cooperation. In Vijay Atluri and Angelos D. Keromytis, editors, *WORM*, pages 72–80. ACM Press, 2005.

- [56] Damon Mosk-Aoyama and Devavrat Shah. Fast distributed algorithms for computing separable functions. *IEEE Transactions on Information Theory*, 54(7):2997–3007, 2008.
- [57] Moni Naor and Udi Wieder. A simple fault tolerant distributed hash table. In *IPTPS*, pages 88–97, 2003.
- [58] Regina O’Dell and Roger Wattenhofer. Information dissemination in highly dynamic graphs. In *DIALM-POMC*, pages 104–110, 2005.
- [59] Gopal Pandurangan, Prabhakar Raghavan, and Eli Upfal. Building low-diameter P2P networks. In *FOCS*, pages 492–499, 2001.
- [60] Gopal Pandurangan, Peter Robinson, and Amitabh Trehan. Dex: Self-healing expanders. In *IEEE IPDPS*, 2014.
- [61] Gopal Pandurangan and Amitabh Trehan. Xheal: localized self-healing using expanders. In *PODC*, pages 301–310, 2011.
- [62] Christian Scheideler. How to spread adversarial nodes?: rotate! In *STOC*, pages 704–713, 2005.
- [63] Christian Scheideler and Stefan Schmid. A distributed and oblivious heap. In *Automata, Languages and Programming*, volume 5556 of *Lecture Notes in Computer Science*, pages 571–582. Springer Berlin / Heidelberg, 2009.
- [64] Subhabrata Sen and Jia Wang. Analyzing peer-to-peer traffic across large networks. IMW ’02, pages 137–150, New York, NY, USA, 2002. ACM.
- [65] Daniel Stutzbach and Reza Rejaie. Understanding churn in peer-to-peer networks. IMC ’06, pages 189–202, New York, NY, USA, 2006. ACM.
- [66] Eli Upfal. Tolerating a linear number of faults in networks of bounded degree. *Inf. Comput.*, 115(2):312–320, 1994.
- [67] Vasileios Vlachos, Stephanos Androutsellis-Theotokis, and Diomidis Spinellis. Security applications of peer-to-peer networks. *Comput. Netw.*, 45:195–205, June 2004.
- [68] Maxwell Young, Aniket Kate, Ian Goldberg, and Martin Karsten. Practical Robust Communication in DHTs Tolerating a Byzantine Adversary. In *ICDCS*, pages 263–272, 2010.