Distributed Computation in Dynamic Networks

Fabian Kuhn
Faculty of Informatics,
University of Lugano
Lugano, Switzerland 6904
fabian.kuhn@usi.ch

Nancy Lynch
Computer Science and AI
Laboratory, MIT
Cambridge, MA 02139
lynch@csail.mit.edu

Rotem Oshman
Computer Science and AI
Laboratory, MIT
Cambridge, MA 02139
rotem@csail.mit.edu

ABSTRACT

In this paper we investigate distributed computation in dynamic networks in which the network topology changes from round to round. We consider a worst-case model in which the communication links for each round are chosen by an adversary, and nodes do not know who their neighbors for the current round are before they broadcast their messages. The model captures mobile networks and wireless networks, in which mobility and interference render communication unpredictable. In contrast to much of the existing work on dynamic networks, we do not assume that the network eventually stops changing; we require correctness and termination even in networks that change continually. We introduce a stability property called T*-interval connectivity* (for $T \geq 1$), which stipulates that for every T consecutive rounds there exists a stable connected spanning subgraph. For $T = 1$ this means that the graph is connected in every round, but changes arbitrarily between rounds.

We show that in 1-interval connected graphs it is possible for nodes to determine the size of the network and compute any computable function of their initial inputs in $O(n^2)$ rounds using messages of size $O(\log n + d)$, where d is the size of the input to a single node. Further, if the graph is T-interval connected for $T > 1$, the computation can be sped up by a factor of T, and any function can be computed in $O(n + n^2/T)$ rounds using messages of size $O(\log n + d)$. We also give two lower bounds on the token dissemination problem, which requires the nodes to disseminate k pieces of information to all the nodes in the network.

The T-interval connected dynamic graph model is a novel model, which we believe opens new avenues for research in the theory of distributed computing in wireless, mobile and dynamic networks.

Categories and Subject Descriptors:
F.2.2 [Analysis of Algorithms and Problem Complexity]: Non-numerical Algorithms and Problems—*computations on discrete structures*
G.2.2 [Discrete Mathematics]: Graph Theory—*graph algorithms*
G.2.2 [Discrete Mathematics]: Graph Theory—*network problems*

General Terms: Algorithms, Theory

Keywords: distributed algorithms, dynamic networks

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
STOC'10, June 5–8, 2010, Cambridge, Massachusetts, USA.
Copyright 2010 ACM 978-1-4503-0050-6/10/06 ...$10.00.

1. INTRODUCTION

The study of dynamic networks has gained importance and popularity over the last few years. Driven by the growing ubiquity of the Internet and a plethora of mobile devices with communication capabilities, novel distributed systems and applications are now within reach. The networks in which these applications must operate are inherently dynamic; typically we think of them as being large and completely decentralized, so that each node can have an accurate view of only its local vicinity. Such networks change over time, as nodes join, leave, and move around, and as communication links appear and disappear.

In some networks, e.g., peer-to-peer, nodes participate only for a short period of time, and the topology can change at a high rate. In wireless ad-hoc networks, nodes are mobile and move around unpredictably. Much work has gone into developing algorithms that are guaranteed to work in networks that eventually stabilize and stop changing; this abstraction is unsuitable for reasoning about truly dynamic networks.

The objective of this paper is to make a step towards understanding the fundamental possibilities and limitations for distributed algorithms in dynamic networks in which eventual stabilization of the network is not assumed. We introduce a general dynamic network model, and study computability and complexity of essential, basic distributed tasks. Under what conditions is it possible to elect a leader or to compute an accurate estimate of the size of the system? How efficiently can information be disseminated reliably in the network? To what extent does stability in the communication graph help solve these problems? These and similar questions are the focus of our current work.

1.1 The Dynamic Graph Model

In the interest of broad applicability our dynamic network model makes few assumptions about the behavior of the network, and we study it from the worst-case perspective. In the current paper we consider a fixed set of nodes that operate in synchronized rounds and communicate by broadcast. In each round the communication graph is chosen adversarially, under an assumption of T*-interval connectivity*: throughout every block of T consecutive rounds there must exist a connected spanning subgraph that remains stable.

We consider the range from 1-interval connectivity, in which the communication graph can change completely from one round to the next, to ∞-interval connectivity, in which there exists some stable connected spanning subgraph that is not known to the nodes in advance. We note that edges that do not belong to the stable subgraph can still change arbitrarily from one round to the next, and nodes do not know which edges are stable and which are not. We do not assume that a neighbor-discovery mechanism is available to

the nodes; they have no means of knowing ahead of time which nodes will receive their message.

In this paper we are mostly concerned with deterministic algorithms, but we also include a randomized algorithm and a randomized lower bound. The computation model is as follows. In every round, the adversary first chooses the edges for the round; for this choice it can see the nodes' internal states at the beginning of the round. At the same time and independent of the adversary's choice of edges, each node tosses private coins and uses them to generate its message for the current round. Deterministic algorithms generate the message based on the interal state alone. In both cases the nodes do not know which edges were chosen by the advesary. Each message is then delivered to the sender's neighbors, as chosen by the adversary; the nodes transition to new states, and the next round begins. Communication is assumed to be bidirectional, but this is not essential. We typically assume that nodes initially know nothing about the network, and communication is limited to $O(\log n)$ bits per message.

To demonstrate the power of the adversary in the dynamic graph model, consider the problem of *local token circulation*: each node u has a local Boolean variable $token_u$, and if $token_u = 1$, node u is said to "have the token". In every round exactly one node in the network has the token, and it can either keep the token or pass it to one of its neighbors. The goal is for all nodes to eventually have the token in some round. This problem is impossible to solve in 1-interval connected graphs: in every round, the adversary can see which node u has the token, and provide that node with only one edge $\{u, v\}$. Node u then has no choice except to eventually pass the token to v. After v receives it, the adversary can turn around and remove all of v's edges except $\{u, v\}$, so that v has no choice except to pass the token back to u. In this way the adversary can prevent the token from ever visiting any node except u, v.

Perhaps surprisingly given our powerful adversary, even in 1-interval connected graphs it is possible to reliably compute any computable function of the initial states of the nodes, and even have all nodes output the result at the same time (simultaneity).

The dynamic graph model we suggest can be used to model various dynamic networks. Perhaps the most natural scenario is mobile networks, in which communication is unpredictable due to the mobility of the agents. There is work on achieving continual connectivity of the communication graph in this setting (e.g., [14]), but currently little is known about how to take advantage of such a service. The dynamic graph model can also serve as an abstraction for static or dynamic wireless networks, in which collisions and interference make it difficult to predict which messages will be delivered, and when. Finally, dynamic graphs can be used to model traditional communication networks, replacing the traditional assumption of a bounded number of failures with our connectivity assumption.

Although we assume that the node set is static, this is not a fundamental limitation. We defer in-depth discussion to future work; however, our techniques are amenable to standard methods such as logical time, which could be used to define the permissible outputs for a computation with a dynamic set of participants.

1.2 Contribution

In this paper we focus on two problems in the context of dynamic graphs. The first problem is *counting*, in which nodes must determine the size of the network. The second is *k-token dissemination*, in which k pieces of information, called *tokens*, are handed out to some nodes in the network, and all nodes must collect all k tokens.

We are especially interested in the variant of k-token dissemination where the number of tokens is equal to the number of nodes in the network, and each node starts with exactly one token. This variant of token dissemination allows any function of the initial states of the nodes to be computed. However, it requires counting, since nodes do not know in advance how many tokens they need to collect. We show that both problems can be solved in $O(n^2)$ rounds in 1-interval connected graphs. Then we extend the algorithm for T-interval connected graphs with known $T > 1$, obtaining an $O(n+n^2/T)$-round protocol for counting or all-to-all token dissemination. When T is not known, we show that both problems can be solved in $O(\min\{n^2, n + n^2 \log(n)/T\})$ rounds. Finally, we give a randomized algorithm for approximate counting that assumes an *oblivious* adversary, and terminates with high probability in almost-linear time.

We also give two lower bounds, both concerning token-forwarding algorithms for token dissemination. A *token-forwarding algorithm* is one that does not combine or alter tokens, only stores and forwards them. First, we give an $\Omega(n \log k)$ lower bound on k-token dissemination in 1-interval connected graphs. This lower bound holds even against centralized algorithms, in which each node is told which token to broadcast by some central authority that can see the entire state of the network. We also give an $\Omega(n + nk/T)$ lower bound on k-token dissemination in T-interval connected graphs for a restricted class of randomized algorithms, in which the nodes' behavior depends only on the set of tokens they knew in each round up to the current one. This includes the algorithms in the paper, and other natural strategies such as round robin, choosing a token to broadcast uniformly at random, or assigning a probability to each token that depends on the order in which the tokens were learned.

For simplicity, the results we present here assume that all nodes start the computation in the same round. It is generally not possible to solve any non-trivial problem if some nodes are initially asleep and do not participate. However, if 2-interval connectivity is assumed, it becomes possible to solve k-token dissemination and counting even when computation is initiated by one node and the rest of the nodes are asleep.

1.3 Related Work

For static networks, information dissemination and basic network aggregation tasks have been extensively studied (see e.g. [5, 20, 34]). In particular, the token dissemination problem is analyzed in [40], where it is shown that k tokens can always be broadcast in time $O(n + k)$ in a static graph. The various problems have also been studied in the context of alternative communication models. A number of papers look at the problem of broadcasting a single message (e.g. [9, 27]) or multiple messages [13, 30] in wireless networks. Gossiping protocols are another style of algorithm in which it is assumed that in each round each node communicates with a small number of randomly-chosen neighbors. Various information dissemination problems for the gossiping model have been considered [21, 23, 25]; gossiping aggregation protocols that can be used to approximate the size of the system are described in [24, 36]. The gossiping model differs from our dynamic graph model in that the neighbors for each node are chosen at random and not adversarially, and in addition, pairwise interaction is usually assumed where we assume broadcast.

A dynamic network topology can arise from node and link failures; fault tolerance, i.e., resilience to a bounded number of faults, has been at the core of distributed computing research from its very beginning [5, 34]. There is also a large body of previous work on general dynamic networks. However, in much of the existing work, topology changes are restricted and assumed to be "well-behaved" in some sense. One popular assumption is eventual sta-

bilization (e.g., [1, 7, 8, 41, 22]), which asserts that changes eventually stop occuring; algorithms for this setting typically guarantee safety throughout the execution, but progress is only guaranteed to occur after the network stabilizes. Self-stabilization is a useful property in this context: it requires that the system converge to a valid configuration from any arbitrary starting state. We refer to [15] for a comprehensive treatment of this topic. Another assumption, studied for example in [26, 28, 35], requires topology changes to be infrequent and spread out over time, so that the system has enough time to recover from a change before the next one occurs. Some of these algorithms use link-reversal [18], an algorithm for maintaining routes in a dynamic topology, as a building block.

Protocols that work in the presence of continual dynamic changes have not been as widely studied. Early work (e.g., [6]) considered the problem of end-to-end message delivery in continually changing networks under an assumption of *eventual connectivity*, which asserts that the source and the destination are connected by a path whose links appear infinitely often during the execution. There is some work on handling nodes that join and leave continually in peer-to-peer overlay networks [19, 31, 33]. Most closely related to the problems studied here is [37], where a few basic results in a similar setting are proved; mainly it is shown that in 1-interval connected dynamic graphs (the definition in [37] is slightly different), if nodes have unique identifiers, it is possible to globally broadcast a single message and have all nodes eventually stop sending messages. The time complexity is at least linear in the value of the largest node identifier. In [2], Afek and Hendler give lower bounds on the message complexity of global computation in asynchronous networks with arbitrary link failures.

The time required for global broadcast has been studied in a probabilistic version of the edge-dynamic graph model, where edges are independently formed and removed according to simple Markov processes [10, 11, 12]. Similar edge-dynamic graphs have also been considered in control theory literature, e.g. [38, 39]. In [12] the authors also consider a worst-case dynamic graph model which is similar to ours, except that the graph is not always connected and collisions are modelled explicitly. This lower-level model does not admit a deterministic algorithm for global broadcast; however, [12] gives a randomized algorithm that succeeds with high probability.

A variant of T-interval connectivity was used in [29], where two of the authors studied clock synchronization in *asynchronous* dynamic networks. In [29] it is assumed that the network satisfies T-interval connectivity for a small value of T, which ensures that a connected subgraph exists long enough for each node to send one message. This is analogous to 1-interval connectivity in synchronous dynamic networks.

Finally, a somewhat related computational model results from population protocols, introduced in [3], where the system is modeled as a collection of finite-state agents with pairwise interactions. Population protocols typically (but not always) rely on a strong fairness assumption which requires every pair of agents to interact infinitely often in an infinite execution. We refer to [4] for a survey. Unlike our work, population protocols compute some function in the limit, and nodes do not know when they are done. This can make sequential composition of protocols challenging, since it is not possible to execute a protocol until it terminates, then take the final result and use it as input for some other computation. (Instead, one may use *self-stabilizing* population protocols, which are resilient to inputs that fluctuate and eventually stabilize to some value; but this is not always possible [16]). In our model nodes must eventually output the result of the computation, and sequential composition is straightforward.

2. PRELIMINARIES

We assume that nodes have unique identifiers (UIDs) drawn from a namespace $\mathcal{U}$. We use $x_u(r)$ to denote the value of node u's local variable x at the beginning of round r.

A synchronous dynamic network is modeled as a dynamic graph $G = (V, E)$, where V is a static set of nodes, and $E : \mathbb{N} \to \{\{u, v\} \mid u, v \in V\}$ is a function mapping a round number $r \in \mathbb{N}$ to a set of undirected edges $E(r)$. We make the following assumption about connectivity in the network graph.

DEFINITION 2.1 (T-INTERVAL CONNECTIVITY). *We say a dynamic graph $G = (V, E)$ is T-*interval connected *for $T \geq 1$ if for all $r \in \mathbb{N}$, the static graph $G_{r,T} := (V, \bigcap_{i=r}^{r+T-1} E(r))$ is connected. The graph is said to be ∞-*interval connected *if there is a connected static graph $G' = (V, E')$ such that for all $r \in \mathbb{N}$, $E' \subseteq E(r)$.*

For the current paper we are mainly interested in the following problems.

Counting. An algorithm is said to solve the counting problem if whenever it is executed in a dynamic graph comprising n nodes, all nodes eventually terminate and output n.

k-verification. Closely related to counting, the k-verification problem requires nodes to determine whether or not $n \leq k$. All nodes begin with k as their input, and must eventually terminate and output "yes" or "no". Nodes must output "yes" if and only if there are at most k nodes in the network.

k-token dissemination. An instance of k-token dissemination is a pair (V, I), where $I : V \to \mathcal{P}(\mathcal{T})$ assigns a set of tokens from some domain $\mathcal{T}$ to each node, and $|\bigcup_{u \in V} I(v)| = k$. An algorithm solves k-token dissemination if for all instances (V, I), when the algorithm is executed in any dynamic graph $G = (V, E)$, all nodes eventually terminate and output $\bigcup_{u \in V} I(u)$. We assume that each token in the nodes' input is represented using $O(\log n)$ bits. Nodes may or may not know k, depending on the context.

All-to-all token dissemination. A restricted class of k-token dissemination in which $k = n$ and for all $u \in V$ we have $|I(u)| = 1$. The nodes know that each node starts with a unique token, but they do not know n.

k-committee election. As a useful step towards solving counting and token dissemination we introduce a new problem called k-committee election. In this problem, nodes must partition themselves into sets, called *committees*, such that

(a) The size of each committee is at most k, and
(b) If $k \geq n$, then there is just one committee containing all nodes.

Each committee has a unique committee ID, and the goal is for all nodes to eventually output a committee ID such that the two conditions are satisfied.

3. BASIC FACTS

In this section we state several basic properties of the dynamic graph model, which we later use in our algorithms. The first key fact pertains to the way information spreads in connected dynamic networks.

PROPOSITION 3.1. *It is possible to solve 1-token dissemination in 1-interval connected graphs in $n - 1$ rounds, if nodes are not required to halt after they output the token.*[1]

[1]Prop. 3.1 is intended only as an illustration; in the rest of our algorithms nodes can halt after they perform the output action.

PROOF SKETCH. We simply have all nodes that know the token broadcast it in every round; when a node receives the token, it outputs it immediately, but continues broadcasting it. In any given round, consider a cut between the nodes that already received the token and those that have not. From 1-interval connectivity, there is an edge in the cut; the token is broadcast on that edge and some new node receives it. Since one node initially knows the message and there are n nodes, after $n-1$ rounds all nodes have the token. □

To make this intuition more formal we use Lamport's causal order [32], which formalizes the notion of one node "influencing" another through a chain of messages originating at the first node and ending at the latter (possibly going through other nodes in between). We use $(u,r) \rightsquigarrow (v,r')$ to denote the fact that node u's state in round r influences node v's state in round r', and the formal definition is as follows.

DEFINITION 3.1 (LAMPORT CAUSALITY). *Given a dynamic graph $G = (V,E)$ we define an order $\rightarrow \subseteq (V \times \mathbb{N})^2$, where $(u,r) \rightarrow (v,r')$ iff $r' = r+1$ and $\{u,v\} \in E(r)$. The* causal order $\rightsquigarrow \subseteq (V \times \mathbb{N})^2$ *is defined to be the reflexive and transitive closure of $\rightarrow$.*

The following lemma shows that 1-interval connectivity is sufficient to guarantee that the number of nodes that have influenced a node u grows by at least one in every round, and so does the number of nodes that u itself has influenced.

LEMMA 3.2. *For any node $u \in V$ and round $r \geq 0$ we have*

(a) $|\{v \in V : (u,0) \rightsquigarrow (v,r)\}| \geq \min\{r+1, n\}$, *and*

(b) $|\{v \in V : (v,0) \rightsquigarrow (u,r)\}| \geq \min\{r+1, n\}$.

The proof of the lemma is similar to that of Proposition 3.1, and it is omitted here. We can now re-state the principle behind Proposition 3.1 as a corollary of Lemma 3.2.

COROLLARY 3.3. *For all $u, v \in V$ it holds that $(v,0) \rightsquigarrow (u, n-1)$.*

PROOF. Lemma 3.2 shows that in round $r = n-1$ we have $|\{v \in V : (v,0) \rightsquigarrow (u,n-1)\}| \geq n$, and the claim follows. □

If we have an upper bound on the size of the network, we can use Corollary 3.3 to compute simple functions which serve as building blocks for algorithms.

PROPOSITION 3.4. *Given an upper bound N on the size of the network, functions such as the minimum or maximum of inputs to the nodes can be computed in $N-1$ rounds.*

Corollary 3.3 guarantees that if nodes always broadcast the smallest (resp. largest) value they have heard, all nodes will have the true min or max value after $n-1$ rounds; the upper bound N is needed for nodes to *know* when they have the true min or max. One application is leader election, which can be implemented by choosing the node with the smallest UID as the unique leader. We also note that having an upper bound on the size allows the use of randomized algorithms for data aggregation which rely on computing the max or the min of random variables chosen by the nodes [17, 36]; see Section 7.

The remainder of the paper focuses on counting and solving the token dissemination problem. The two problems are intertwined, and both are useful as a starting point for distributed computing in dynamic networks. We remark that when message sizes are not limited, both problems can be solved in linear time by having nodes constantly broadcast all the information they have collected so far.

PROPOSITION 3.5. *Counting and all-to-all token dissemination can be solved in $O(n)$ rounds in 1-interval connected graphs, using messages of size $O(n \log n)$.*

PROOF. Consider a simple protocol for counting. Each node maintains a set A containing all the UIDs it has heard about so far, where initially $A_u(0) = \{u\}$ for all $u \in V$. In each round r, node u broadcasts A_u and adds to A_u any UIDs it receives from other nodes. If $r > |A_u|$, node u halts and outputs $|A_u|$; otherwise node u continues on to the next round.

It is not hard to see that for all $u, v \in V$ and rounds r, if $(v,0) \rightsquigarrow (u,r)$ then $v \in A_u(r)$. Thus, $\{v \in V : (v,0) \rightsquigarrow (u,r)\} \subseteq A_u(r)$. Correctness of the protocol follows from Lemma 3.2: if node u halts in round r, then $r > |A_u(r)| \geq |\{v \in V : (v,0) \rightsquigarrow (u,r)\}|$, and Lemma 3.2 shows that $r > n$. Next, using Corollary 3.3 we have that in this case $V \subseteq A_u(r)$. And finally, since obviously $A_u(r) \subseteq V$, it follows that $A_u(r) = V$ and node u's output is correct. Termination also follows from Lemma 3.2 and the fact that $A_u(r) \subseteq V$ in every round r.

To solve all-to-all token dissemination, we have nodes attach every token they have heard so far to every message they send. □

In the sequel we describe solutions which use only $O(\log n)$ bits per message.

4. COUNTING THROUGH k-COMMITTEE

In this section we show how k-committee election can be used to solve counting and token dissemination.

Our counting algorithm works by successive doubling: at each point the nodes have a guess k for the size of the network, and attempt to verify whether or not $k \geq n$. If it is discovered that $k < n$, the nodes double k and repeat; if $k \geq n$, the nodes halt and output the count. We defer the problem of determining the exact count until the end of the section, and focus for now on the k-verification problem, that is, checking whether or not $k \geq n$.

Suppose that nodes start out in a state that represents a solution to k-committee election: each node has a committee ID, such that no more than k nodes have the same ID, and if $k \geq n$ then all nodes have the same committee ID. The problem of checking whether $k \geq n$ is then equivalent to checking whether there is more than one committee: if $k \geq n$ there must be one committee only, and if $k < n$ there must be more than one. Nodes can therefore check if $k \geq n$ by executing a simple k-round protocol that checks if there is more than one committee in the graph.

The k-verification protocol. Each node has a local variable x, which is initially set to 1. While $x_u = 1$, node u broadcasts its committee ID. If it hears from some neighbor a different committee ID from its own, or the special value $\perp$, it sets $x_u \leftarrow 0$ and broadcasts $\perp$ in all subsequent rounds. After k rounds, all nodes output the value of their x variable.

LEMMA 4.1. *If the initial state of the execution represents a solution to k-committee election, at the end of the k-verification protocol each node outputs 1 iff $k \geq n$.*

PROOF SKETCH. First suppose that $k \geq n$. In this case there is only one committee in the graph; no node ever hears a different committee ID from its own. After k rounds all nodes still have $x = 1$, and all output 1.

Next, suppose $k < n$. We can show that after the ith round of the protocol, at least i nodes in each committee have $x = 0$. In any round of the protocol, consider a cut between the nodes that belong to a particular committee and still have $x = 1$, and the rest

of the nodes, which either belong to a different committee or have $x = 0$. From 1-interval connectivity, there is an edge in the cut, and some node u in the committee that still has $x_u = 1$ hears either a different committee ID or $\perp$. Node u then sets $x_u \leftarrow 0$, and the number of nodes in the committee that still have $x = 1$ decreases by at least one. Since each committee initially contains at most k nodes, after k rounds all nodes in all committees have $x = 0$, and all output 0. □

Our strategy for solving the counting problem is as follows: for $k = 1, 2, 4, 8, \ldots$, solve the k-committee election problem, then execute the k-verification protocol. If $k \geq n$, terminate and output the count; else, continue to the next value of k. Here we use the fact that our model is amenable to sequential composition.

The strategy outlined above requires all nodes to begin the k-verification protocol in the same round. Our protocol for solving k-committee election ensures that this occurs. The protocol also has the useful property that if $k \geq n$, every node knows the UIDs of all other nodes in the graph at the end of the protocol. Thus, when $k \geq n$, nodes can determine the exact count.

5. A k-COMMITTEE PROTOCOL FOR 1-INTERVAL CONNECTED GRAPHS

To solve k-committee election in 1-interval connected graphs, we imagine that there is a unique leader in the network, and this leader invites k nodes to join its committee. Of course we do not truly have a pre-elected leader in the network; we will soon show how to get around this problem. The protocol proceeds in k cycles, each consisting of two phases.

- **Polling phase:** For $k - 1$ rounds, all nodes in the network propagate the UID of the smallest node they have heard about that has not yet joined a committee. Initially each node broadcasts its own UID if it has not joined a committee, or $\perp$ if it has; in each round nodes remember the smallest value they have sent or received so far in the execution, and broadcast that value in the next round.
- **Invitation phase:** The leader selects the smallest UID it heard during the polling phase, and issues a message inviting that node to join its committee. The message carries the UID of the leader and of the invited node. The invitation is propagated by all nodes for $k - 1$ rounds. At the end of the invitation phase, a node that received an invitation joins the leader's committee.

At the end of the k cycles, nodes that have joined the leader's committee output the leader's UID as their committee ID. Any node that has not been invited to join a committee joins its own committee, using its UID as the committee ID.

Because we do not initially have a unique leader in the network, *all* nodes start out thinking they are the leader, and continue to play the role of a leader until they hear a UID smaller than their own. At that point they switch to playing the role of a non-leader. However, once nodes join a committee they do not change their minds.

THEOREM 5.1. *The protocol sketched above solves k-committee election in $O(k^2)$ rounds.*

PROOF SKETCH. The first condition of k-committee election requires each committee to be of size at most k. This condition is satisfied because no node ever invites more than k nodes to join its committee (each node issues at most one invitation per cycle). For the second condition we must show that if $k \geq n$ then all nodes join the same committee. Thus, suppose that $k \geq n$. The polling phase of the first cycles lasts for $k - 1 \geq n - 1$ rounds, and from Corollary 3.3, this is sufficient for all nodes to hear the UID of the smallest node in the network. Thus, after the first polling phase there is only one leader, and no other node ever issues an invitation.

Using Corollary 3.3 we see that the $k - 1$ rounds of each polling phase are sufficient for the leader to successfully identify the smallest node that has not yet joined its committee. Similarly, the invitation phase is long enough for that node to receive the leader's invitation, so in every cycle one node joins the leader's committee. Since there are $k \geq n$ cycles, all nodes join the leader's committee, and all output the leader's UID as their committee ID. □

We remark that when $k \geq n$, the k-committee election protocol can also be used to solve all-to-all token dissemination. To do so we simply have nodes attach their token to their UID in every message they send. Each node is "singled out" for $k - 1 \geq n - 1$ rounds during which it is invited to join the leader's committee, and the invitation reaches all nodes in the graph. Thus, nodes can collect all the tokens by recording the tokens attached to all invitations they hear. In particular, if node UIDs are used as tokens, nodes can collect all the UIDs in the network.

COROLLARY 5.2. *When used together with the k-verification protocol from Section 4, the k-committee election protocol yields an $O(n^2)$-round protocol for counting or all-to-all token dissemination.*

6. COUNTING AND TOKEN DISSEMINATION IN MORE STABLE GRAPHS

In this section we show that in T-interval connected graphs the computation can be sped up by a factor of T. To do this we employ a neat pipelining effect, using the temporarily stable subgraphs that T-interval connectivity guarantees; this allows us to disseminate information more quickly. For convenience we assume that the graph is $2T$-interval connected for some $T \geq 1$.

6.1 Fast T-Token Dissemination in $2T$-Interval Connected Graphs

Procedure `disseminate` gives an algorithm for exchanging at least T pieces of information in n rounds when the dynamic graph is $2T$-interval connected. The procedure takes three arguments: a set of tokens A, the parameter T, and a guess k for the size of the graph. If $k \geq n$, each node is guaranteed to learn the T smallest tokens that appeared in the input to all the nodes.

The execution of procedure `disseminate` is divided into $\lceil k/T \rceil$ phases, each consisting of $2T$ rounds. During each phase, each node maintains the set A of tokens it has already learned and a set S of tokens it has already broadcast in the current phase (initially empty). In each round of the phase, the node broadcasts the smallest token it has not yet broadcast in the current phase, then adds that token to S.

```
S ← ∅
for i = 0, ..., ⌈k/T⌉ − 1 do
    for r = 0, ..., 2T − 1 do
        if S ≠ A then
            t ← min (A \ S)
            broadcast t
            S ← S ∪ {t}
        receive t_1, ..., t_s from neighbors
        A ← A ∪ {t_1, ..., t_s}
    S ← ∅
return A
```

Procedure `disseminate`(A, T, k)

Because the graph is $2T$-interval connected, in each phase i there is a stable connected subgraph G_i that persists throughout the phase. We use $A_u^i(r), S_u^i(r)$ for the values of node u's local variables A, S at the beginning of round r of phase i. We say that u *knows* token t whenever $t \in A_u$.

Let $K_i(t)$ denote the set of nodes that know t at the beginning of phase i, and let $\mathrm{tdist}_i(u,t)$ denote the minimal distance in G_i between node u and any node in $K_i(t)$. Correctness hinges on the following property.

LEMMA 6.1. *For any node $u \in V$, token $t \in \bigcup_{v \in V} A_v(0)$ and round r such that $\mathrm{tdist}_i(u,t) \le r \le 2T$, either $t \in S_u^i(r+1)$ or $S_u(r+1)$ includes at least $(r - \mathrm{tdist}_i(u,t))$ tokens that are smaller than t.*

The intuition behind Lemma 6.1 is that if $r \ge \mathrm{tdist}_i(u,t)$, then r rounds are "enough time" for u to receive t. If u has not received t and sent it on, the path between u and the nearest node that knows t must have been blocked by smaller tokens, which node u received and sent on.

Using Lemma 6.1 we can show:

LEMMA 6.2. *If $k \ge n$, at the end of procedure* `disseminate` *the set A_u of each node u contains the T smallest tokens.*

PROOF SKETCH. Let $N_i^d(t) := \{u \in V \mid \mathrm{tdist}_i(u,t) \le d\}$ denote the set of nodes at distance at most d from some node that knows t at the beginning of phase i, and let t be one of the T smallest tokens.

From Lemma 6.1, for each node $u \in N_i^T(t)$, either $t \in S_u^i(2T+1)$ or $S_u^i(2T+1)$ contains at least $2T - T = T$ tokens that are smaller than t. But t is one of the T smallest tokens, so the second case is impossible. Therefore all nodes in $N_i^T(t)$ know token t at the end of phase i. Because G_i is connected we have $|N_i^T(t)| \ge \min\{n - |K_i(t)|, T\}$; that is, in each phase T new nodes learn t, until all the nodes know t. Since there are no more than k nodes and we have $\lceil k/T \rceil$ phases, at the end of the last phase all nodes know t. □

Remark 1. If each stable subgraph G_i enjoys good expansion then `disseminate` requires fewer than n phases. For example, if G_i is always f-connected for some parameter f, then each token is learned by $f \cdot T$ new nodes in each phase until all nodes know it, and we only require $\lceil n/f \rceil$ phases. Similarly, if G_i is always a vertex expander we only require $O(\log n)$ phases.

6.2 Counting and Token Dissemination

To solve counting and token dissemination with up to n tokens, we use Procedure `disseminate` to speed up the k-committee election protocol from Section 5. Instead of inviting one node in each cycle, we can use `disseminate` to have the leader learn the UIDs of the T smallest nodes in the polling phase, and use procedure `disseminate` again to extend invitations to all T smallest nodes in the selection phase. Thus, in $O(k+T)$ rounds we can increase the size of the committee by T.

THEOREM 6.3. *It is possible to solve k-committee election in $O(k + k^2/T)$ rounds in T-interval connected graphs. When used in conjunction with the k-verification protocol, this approach yields an $O(n+n^2/T)$-round counting all-to-all token dissemination protocol.*

6.3 Unknown Interval Connectivity

The protocol sketched above assumes that all nodes know the degree of interval connectivity present in the communication graph; if the graph is not $2T$-interval connected, invitations may not reach their destination, and the committees formed may contain less than k nodes even when $k \ge n$. However, even when the graph is not $2T$-interval connected, no committee ever contains *more* than k nodes, simply because no node ever issues more than k invitations. Thus, if nodes guess a value for T and use the protocol to check if $k \ge n$, their error is one-sided: if their guess for T is too large they may falsely conclude that $k < n$ when in fact $k \ge n$, but they will never conclude that $k \ge n$ when $k < n$.

This one-sided error allows us to try different values for k and T without fear of mistakes. We can count in $O(n \log n + n^2 \log n/T)$ time in graphs where T is *unknown* by iterating over various combinations of k and T until we reach a pair (k,T) such that $k \ge n$ and the graph is T-interval connected.

In the worst case, the graph is 1-interval connected, and we need to try all the values $T = 1, 2, 4, \ldots, k$ for each k; we pay a $\log n$ factor in the round complexity. This only improves upon the original $O(n^2)$ algorithm when the graph is $\omega(\log n)$-interval connected. However, we can execute the original algorithm in parallel with the adaptive one, and terminate when the first of the two terminates. In this way we can solve counting or token dissemination in $O(\min\{n^2, n \log n + n^2 \log n/T\})$ rounds when T is unknown.

Using similar ideas we can also adapt to unknown expansion of the graph, e.g., we might guess that it is always f-connected for some initial value of f, and decrease f until we find the right value.

7. APPROXIMATE COUNTING

In this section we show that under certain restrictions on the dynamic-graph adversary, it is possible to use randomization to compute an approximate count in almost-linear time, even when the dynamic graph is only 1-interval connected. The techniques we use are based on a gossiping protocol from [36]. We assume that nodes know some (potentially loose) upper bound N on the size n of the network; this upper bound determines the message size.

For any $\varepsilon > 0$, the algorithm computes a $(1+\varepsilon)$-approximation of the number of nodes n. There are two variants of the algorithm: the first terminates in $O(n)$ time with high probability (in N) and uses messages of size $O(\log N \cdot (\log\log N + \log(1/\varepsilon)/\varepsilon^2))$; the second requires $O(n \cdot (\log\log N + \log(1/\varepsilon)/\varepsilon^2))$ rounds with high probability, but uses messages of size only $O(\log N)$.

Both versions of the algorithm assume that the dynamic graph is generated by an *oblivious* adversary, which determines the complete sequence of graphs before the execution begins. In particular, the adversary is not privy to the results of the nodes' coin tosses in previous rounds, and it also cannot see their states and their messages, which reveal the results of those coin tosses.

For simplicity, we describe here only the algorithm that runs in $O(n)$ rounds w.h.p. but uses slightly larger messages.

The algorithm relies on the following lemma from [36], which shows how the size of the network can be estimated by computing the minimum of exponential random variables (and repeating this procedure to decrease the error probability).

LEMMA 7.1 ([36]). *Let S be a set of ℓ-tuples of independent exponential variables with rate 1: $S = \left\{\left(Y_1^{(1)}, \ldots, Y_\ell^{(1)}\right), \ldots, \left(Y_1^{(m)}, \ldots, Y_\ell^{(m)}\right)\right\}$. Define*

$$\hat{n}(S) := \frac{\ell}{\sum_{i=1}^{\ell} \min_{1 \le j \le |S|} Y_i^{(j)}}.$$

Then

$$\Pr\left(\left|\hat{n}(S) - |S|\right| > \frac{2}{3}\epsilon \cdot |S|\right) \leq 2e^{-\varepsilon^2 \ell/27}.$$

For parameters $\varepsilon \in (0, 1/2)$ and $c > 0$, we define $\ell := \lceil (2 + 2c) \cdot 27 \ln(N)/\varepsilon^2 \rceil$.

The scheme for approximate counting is given in Alg 2. The main idea is as follows. Initially, each node $v \in V$ computes ℓ independent exponential random variables $Y_1^{(v)}, \ldots, Y_\ell^{(v)}$ with rate 1. The objective of all nodes is to compute $\hat{n}(V)$, which is a good estimate for n with high probability. To do this they must compute $\min_{v \in V} Y_i^{(v)}$ for each $i \in [\ell]$.

From Proposition 3.4 we know that nodes can find $\min_{v \in V} Y_i^{(v)}$ by propagating the smallest value they have heard so far for $n - 1$ rounds. However, n is not known to the nodes (we could wait $N - 1$ rounds, but N may be a very loose upper bound). We use a combination of Lemma 3.2 and Lemma 7.1 to decide when to stop.

Let $C_u(r) := \{v \in V : (v, 0) \rightsquigarrow (u, r)\}$ be the set of nodes whose value $Y_i^{(v)}$ has reached u by round r. In round r node u is able to compute $\min_{v \in C_u(r)} Y_i^{(v)}$, but it cannot "see" values $Y_i^{(v)}$ for $v \notin C_u(r)$. Therefore we want node u to halt only when $C_u(r) = V$.

From Lemma 3.2 we know that $|C_u(r)| \geq r + 1$ for all $r \leq n - 1$. Because we assume an oblivious adversary, the set $C_u(r)$ is chosen before the nodes choose their random variables. We can use Lemma 7.1 to show that with high probability, if $(1 - \varepsilon)r > \hat{n}_u(C_u(r))$, then $C_u(r) = V$. We use this criterion to know when to terminate. (Recall that in Proposition 3.5 we used a deterministic version of this test: we halted exactly when $r > |C_u(r)|$.)

Sending exact values for $Y_i^{(v)}$ would require nodes to send real numbers, which cannot be represented using a bounded number of bits. Instead nodes send rounded and range-restricted approximations $\tilde{Y}_i^{(v)}$ for $Y_i^{(v)}$; we omit the technical details here. Each value $\tilde{Y}_i^{(v)}$ can be represented using $O(\log \log N + \log(1/\varepsilon))$ bits.

$\underline{Z}^{(u)} \leftarrow (\tilde{Y}_1^{(u)}, \ldots, \tilde{Y}_\ell^{(u)})$
for $r = 1, 2, \ldots$ **do**
  broadcast $\underline{Z}^{(u)}$
  receive $\underline{Z}^{(v_1)}, \ldots, \underline{Z}^{(v_s)}$ from neighbors
  for $i = 1, \ldots, \ell$ **do**
    $Z_i^{(u)} \leftarrow \min\left\{Z_i^{(u)}, Z_i^{(v_1)}, \ldots, Z_i^{(v_s)}\right\}$
  $\tilde{n}_u(r) \leftarrow \ell / \sum_{i=1}^{\ell} Z_i^{(u)}$
  if $(1 - \varepsilon)r > \tilde{n}_u(r)$ **then** terminate and output $\tilde{n}_u(r)$

Algorithm 2: Randomized approximate counting in linear time (code for node u)

For lack of space, the following theorem is given without proof.

THEOREM 7.2. *For $\varepsilon \in (0, 1/2)$ and $c > 0$, with probability at least $1 - 1/N^c$,*

(a) every node in the graph computes the same value $\tilde{n}_v(r) =: \tilde{n}$, and furthermore,

(b) $|\tilde{n} - n| \leq \varepsilon n$.

8. LOWER BOUNDS ON TOKEN DISSEMINATION

Our algorithms for token dissemination do not combine tokens or alter them in anyway, only store and forward them. We call this style of algorithm a *token-forwarding algorithm*. Formally, let $A_u(r)$ denote the set of messages node u has received by the beginning of round r, plus node u's input $I(u)$. A token-forwarding algorithm satisfies: (a) for all $u \in V$ and $r \geq 0$, the message sent by u in round r is a member of $A_u(r) \cup \{\perp\}$, where $\perp$ denotes the empty message; and (b) node u cannot halt in round r unless $A_u(r) = \bigcup_{v \in V} I(v)$, that is, node u has received all the tokens either in messages from other nodes or in its input.

In this section we give two lower bounds on token dissemination with token-forwarding algorithms.

8.1 Lower Bound on Centralized Token Dissemination

For this lower bound we assume that in each round r, some central authority provides each node u with a value $t_u(r) \in A_u(r)$ to broadcast in that round. The centralized algorithm can see the state and history of the entire network, but it does not know which edges will be scheduled in the current round. Centralized algorithms are more powerful than distributed ones, since they have access to more information. To simplify, we begin with each of the k tokens known to exactly one node (this restriction is not essential).

We observe that while the nodes only know a small number of tokens, it is easy for the algorithm to make progress; for example, in the first round of the algorithm at least k nodes learn a new token, because connectivity guarantees that k nodes receive a token that was not in their input. However, as nodes learn more tokens, it becomes harder for the algorithm to provide them with tokens they do not already know.

Accordingly, our strategy is to charge a cost of $1/(k - i)$ for the i-th token learned by each node: the first token each node learns comes at a cheap $1/k$, and the last token learned costs dearly (a charge of 1). Formally, the potential of the system in round r is given by

$$\Phi(r) := \sum_{u \in V} \sum_{i=0}^{|A_u(r)|-1} \frac{1}{k - i}.$$

In the first round we have $\Phi(0) = 1$, because k nodes know one token each. If the algorithm terminates in round r then we must have $\Phi(r) = n \cdot H_k = \Theta(n \log k)$, because all n nodes must know all k tokens. We construct an execution in which the potential increase is bounded by a constant in every round; this gives us an $\Omega(n \log k)$ bound on the number of rounds required.

THEOREM 8.1. *Any deterministic centralized algorithm for k-token dissemination in 1-interval connected graphs requires at least $\Omega(n \log k)$ rounds to complete in the worst case.*

PROOF. We construct the communication graph for each round r in three stages (independently of previous or future rounds).

Stage I: adding the free edges. An edge $\{u, v\}$ is said to be *free* if $t_u(r) \in A_v(r)$ and $t_v(r) \in A_u(r)$; that is, if when we connect u and v, neither node learns anything new. Let $F(r)$ denote the set of free edges in round r; we add all of them to the graph. Let $C_1, \ldots, C_\ell$ denote the connected components of the graph $(V, F(r))$. Observe that any two nodes in different components must send different values, otherwise they would be in the same component.

We choose representatives $v_1 \in C_1, \ldots, v_\ell \in C_\ell$ from each component arbitrarily. Our task now is to construct a connected subgraph over $v_1, \ldots, v_\ell$ and pay only a constant cost. We assume that $\ell \geq 12$, otherwise we can connect the nodes arbitrarily for a constant cost. Let $missing(u) := k - |A_u(r)|$ denote the number of tokens node u does not know at the beginning of round r.

Stage II. We split the nodes into two sets, Top and $Bottom$, according to the number of tokens they know, with nodes that know many tokens "on top": $Top := \{v_i \mid missing(v_i) \le \ell/6\}$ and $Bottom := \{v_i \mid missing(v_i) > \ell/6\}$.

Since top nodes know many tokens, connecting to them could be expensive. We will choose our edges in such a way that no top node will learn a new token. Bottom nodes are cheaper, but still not free; we will ensure that each bottom node will learn at most three new tokens (see Fig. 1).

We begin by bounding the size of Top. To that end, notice that $\sum_{u \in Top} missing(u) \ge \binom{|Top|}{2}$: for all i, j such that $u, v \in Top$, either $t_u(r) \notin A_v(r)$ or $t_v(r) \notin A_u(r)$, otherwise $\{u, v\}$ would be a free edge and u, v would be in the same component. Therefore each pair $u, v \in Top$ contributes at least one missing token to the sum, and $\sum_{u \in Top} missing(u) \ge \binom{|Top|}{2}$. On the other hand, since each node in Top is missing at most $\ell/6$ tokens, it follows that $\sum_{u \in Top} missing(u) \le |Top| \cdot (\ell/6)$. Putting the two facts together we obtain $|Top| \le \ell/3 + 1$, and consequently also

$$|Bottom| - |Top| \ge \ell - 2|Top| \ge \ell - \frac{2\ell}{3} - 2 \overset{(\ell \ge 12)}{\ge} \frac{\ell}{6}.$$

Next we show that because there are many more bottom nodes than top nodes, we have enough flexibility to use only "cheap" edges to connect to top nodes.

Stage III: Connecting the nodes. The bottom nodes are relatively cheap to connect to, so we connect them in an arbitrary line (see Fig. 1). In addition we want to connect each top node to a bottom node, such that no top node learns something new, and no bottom node is connected to more than one top node. That is, we are looking for a matching between Top and $Bottom$, using only edges in $P = \{\{u, v\} : u \in Top, v \in Bottom \text{ and } t_v \in A_u(r)\}$.

Since each top node is missing at most $\ell/6$ tokens, and each bottom node broadcasts a different value from all other bottom nodes, for each top node there are at least $|Bottom| - \ell/6$ edges in P to choose from. To construct the matching, we go through the top nodes in arbitrary order $v_0, \ldots, v_p \in Top$, and choose for each v_i some unmatched bottom node u_i such that $\{v_i, u_i\} \in P$ and $u_i \ne u_j$ for all $j < i$. Before each step i the number of unmatched bottom nodes is at least $|Bottom| - i > |Bottom| - |Top| \ge \ell/6$. We already saw that each top node is connected to all but $\ell/6$ bottom nodes in P, so there is always some unmatched P-neighbor of v_i to choose in step i.

What is the total cost of the graph? Top nodes learn no tokens, and bottom nodes learn at most two tokens from other bottom nodes and at most one token from a top node. Thus, the total cost is bounded by

$$\sum_{u \in Bottom} \sum_{i=1}^{\min\{3, missing(u)\}} \frac{1}{missing(u) - (i-1)}$$
$$\le |Bottom| \cdot \frac{6}{\frac{\ell}{6}} \le \ell \cdot \frac{36}{\ell} = 36.$$

□

8.2 Lower Bound on Token Dissemination with Knowledge-Based Algorithms

A token-forwarding randomized algorithm for k-token dissemination is said to be *knowledge-based* if the distribution that determines which token is broadcast by node u in round r is a function of the UID of u, the sequence $A_u(0), \ldots, A_u(r-1)$, where A_i is the set of tokens received by u by the beginning of round i (including its input), and the sequence of u's coin tosses up to round r (inclusive).

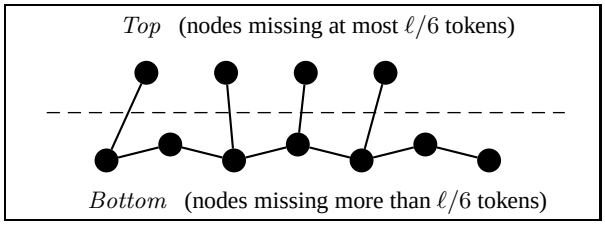


Figure 1: Illustration for the proof of Theorem 8.1

Knowledge-based algorithms can base their decisions on the set of tokens currently known, the order in which tokens were acquired, and even the round in which each token was acquired; however, they cannot rely on other factors, such as the number of times a particular token was heard, or which tokens were received in the previous round. Nevertheless, the class of knowledge-based algorithms includes many natural strategies for solving the token dissemination problem, and it includes the algorithms in this paper. (Other knowledge-based strategies include round-robin over the known tokens, choosing a token to broadcast uniformly at random, and choosing each token with a probability that depends on how long ago that token was acquired.)

Knowledge-based algorithms have the property that once a node learns all the tokens, the distribution of tokens broadcast in future rounds is fixed and does not depend on the dynamic graph. We use this property to show the following lower bound.

THEOREM 8.2. *Any knowledge-based algorithm for k-token dissemination in T-interval connected graphs requires $\Omega(n + nk/T)$ rounds to succeed with probability $1/2$. Further, if the size of the namespace for UIDs $|\mathcal{U}| = \Omega(n^2 k/T)$, then deterministic algorithms require $\Omega(n + nk/T)$ rounds even when each node starts with exactly one token.*

PROOF SKETCH. An $\Omega(n)$ lower bound is trivially demonstrated in a static line graph where some token starts at one end of the line. Thus we assume that $k > 1$. For simplicity, we choose an input assignment in which some node u knows all the tokens, and the other nodes have no tokens.

Let $r_0 = (n-1)(k-1)/(4T) - 2 = \Theta(nk/T)$. We say that a token t is *infrequent* in a given execution if node u broadcasts t less than $(n-1)/(2T)$ times in rounds $0, \ldots, r_0$ of the execution.

Since node u knows all the tokens, its behavior is determined: regardless of the dynamic graph we choose in rounds $0, \ldots, r_0$, the distribution of tokens broadcast by node u in these rounds is fixed. In particular, since $r_0 < (n-1)(k-1)/(4T) - 1$, the linearity of expectation and Markov's inequality show that there is some token t such that in any dynamic graph, token t will be infrequent with probability at least $1/2$. We will construct a specific dynamic graph G in which whenever t is infrequent, the algorithm does not terminate by round r_0. Thus, in the graph we construct, the algorithm requires $\Omega(nk/T)$ rounds w.p. at least $1/2$.

Initially there are $n-1$ nodes that do not know t (all nodes but u). Our goal in constructing G is to ensure that every time node u broadcasts t, at most $2T$ new nodes learn t. Recall that t is said to be infrequent when it is broadcast less than $(n-1)/(2T)$ times by round r_0. Hence, whenever t is infrequent, some node in G has still not learned t by round r_0 and algorithm cannot terminate.

We construct the dynamic graph in phases of two types. When

u has not broadcast t for a while (T rounds to be precise), the network is in a *quiet phase*. A quiet phase extends until the first time u broadcasts t (including that round). During quiet phases the communication graph remains static and comprises two components (see Fig. 2(a)): component U (for "unaware", shown as white nodes in Fig. 2(a)) contains nodes that are guaranteed not to know t, arranged in a line $v_{i_1}, \ldots, v_{i_\ell}$. The first node in the line, v_{i_1}, is connected to node u. (Note that $u \notin U$, because u knows t from the start). The other component, K (for "knowledgeable"), contains the remaining nodes. These nodes may or may not know t, and we connect them to each other arbitrarily. Initially, $K = \{u\}$ and $U = V \setminus \{u\}$, with the nodes in U ordered arbitrarily.

A quiet phase ends immediately after u broadcasts token t. At this point $v_{i_1} \in U$ knows t; if we leave the network static, the nodes in U may forward t to each other, until in $|U| - 1$ rounds all nodes in U know t. Recall that we want to ensure that at most $2T$ nodes learn t after every time u broadcasts it. To contain the propagation of t, we begin an *active phase*.

If we wanted to satisfy only 1-interval connectivity, we would simply move v_{i_1} from U to K and connect u to v_{i_2} instead. This would prevent v_{i_1} from spreading t to other nodes in U, but it violates T-interval connectivity for $T > 1$. In order to move nodes from U to K we need more edges, so that we can remove some without breaking connectivity.

Thus, at the beginning of an active phase we connect u to v_{i_ℓ}, closing the line to form a ring (see Fig. 2(b)). Then we wait for T rounds. Finally, we remove edge $\{v_{i_{2T}}, v_{i_{2T+1}}\}$ (see Fig. 2(c)). This ends the active phase. Note that T-interval connectivity is preserved along the line $v_{i_{2T+1}}, v_{i_{2T+2}}, \ldots, v_{i_\ell}, u, v_{i_1}, \ldots, v_{i_{2T}}$ (shown in bold lines in the figures). This is why we use a ring instead of a line.

At the beginning of an active phase, token t may be known only by nodes $v_{i_1}, \ldots, v_{i_T}$ in the ring. (If the preceding phase was quiet, only node v_{i_1} knows t; if the preceding phase was active more nodes may know t, see below.) An active phase lasts T rounds. During this time, token t propagates in one of two ways:

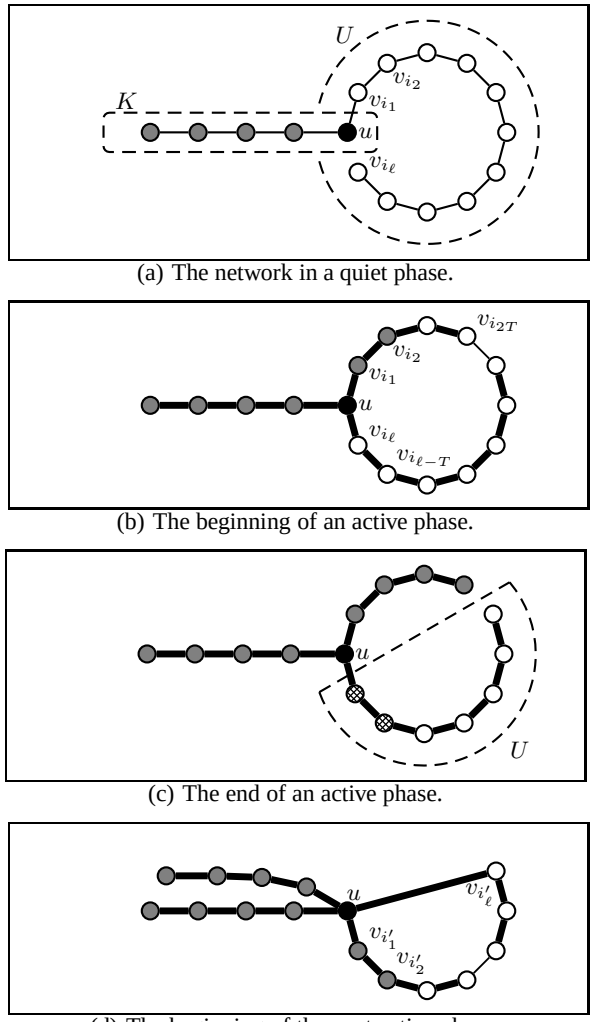
(1) Nodes $v_{i_1}, \ldots, v_{i_T}$ may forward t to nodes $v_{i_{T+1}}, \ldots, v_{i_{2T}}$.

(2) Node u may broadcast t again during the phase, in which case nodes $v_{i_\ell}, \ldots, v_{i_{\ell-T}}$ (indicated in cross-hatching in Fig. 2(c)) may also learn it.

At the end of the phase we remove the link $\{v_{i_{2T}}, v_{i_{2T+1}}\}$, cutting off the propagation of t along that side of the ring, and set $U \leftarrow U \setminus \{v_{i_1}, \ldots, v_{i_{2T}}\}$. Notice that now the only node in U to which u is connected is v_{i_ℓ}. To retain consistency in notation we reverse the line, renaming as follows: $\ell' \leftarrow \ell - 2T, i'_1 \leftarrow i_\ell, i'_2 \leftarrow i_{\ell-1}, \ldots, i'_\ell \leftarrow i_{2T+1}$.

If u did not broadcast t during the phase (that is, case (2) above did not occur), then no remaining node in U knows t, and we begin another quiet phase. If u did broadcast t, at most $v_{i'_1}, \ldots, v_{i'_T}$ (which were labelled $v_\ell, \ldots, v_{\ell-T}$ before the renaming) know it, and we begin another active phase.

The construction above allows us to charge at most $2T$ nodes to each time u broadcasts t: an active phase is only triggered when u broadcasts t, and each active phase ends with the removal of $2T$ nodes from U. If t is infrequent in an execution, then there are less than $(n-1)/(2T)$ active phases by round r_0 of the execution; since initially $|U| = n - 1$, by round $r_0 = \Omega(nk/T)$ there is still some node in U which does not know t, and the algorithm is not done. Since t is infrequent w.p. at least $1/2$, this shows that any knowledge-based algorithm for k-token dissemination requires $\Omega(nk/T)$ rounds w.p. at least $1/2$. □



(a) The network in a quiet phase.

(b) The beginning of an active phase.

(c) The end of an active phase.

(d) The beginning of the next active phase.

Figure 2: The construction for Theorem 8.2, with $T = 2$. Nodes that do not know t are shown in white, nodes that may know t are shown in grey. Edges along which T-interval connectivity is preserved are shown in bold.

9. CONCLUSION

In this work we consider a model for dynamic networks which makes very few assumptions about the network. The model can serve as an abstraction for wireless or mobile networks, to reason about the fundamental unpredictability of communication in this type of system. We do not restrict the mobility of the nodes except for retaining connectivity, and we do not assume that geographical information or neighbor discovery are available. Nevertheless, we show that one can efficiently compute any computable function in our model, taking advantage of stability if it exists in the network.

We believe that the T-interval connectivity property provides a natural and general way to reason about dynamic networks. It is easy to see that without any connectivity assumption no non-trivial function can be computed, except possibly in the sense of computation in the limit (as in [3]). However, our connectivity assumption is easily weakened to only require connectivity once every constant number of rounds, or to only require eventual connectivity in the style of Prop. 3.1, with a known bound on the number of rounds.

There are many open problems related to the model. We hope to strengthen our lower bounds for token dissemination and obtain an $\Omega(nk/T)$ general lower bound, and to determine whether counting is in fact as hard as token dissemination. Other natural problems, such as consensus and leader election, can be solved in linear time once a (possibly approximate) count is known, but can they be solved more quickly without first counting? Is it possible to compute an approximate upper bound for the size of the network in less than the time required for counting exactly? These and other questions remain intriguing open problems.

10. REFERENCES

[1] Y. Afek, B. Awerbuch, and E. Gafni. Applying static network protocols to dynamic networks. In *Proc. of 28th Symp. on Foundations of Computer Science (FOCS)*, pages 358–370, 1987.

[2] Y. Afek and D. Hendler. On the complexity of gloabl computation in the presence of link failures: The general case. *Distributed Computing*, 8(3):115–120, 1995.

[3] D. Angluin, J. Aspnes, Z. Diamadi, M. Fischer, and R. Peralta. Computation in networks of passively mobile finite-state sensors. *Distributed Computing*, 18(4):235–253, 2006.

[4] J. Aspnes and E. Ruppert. An introduction to population protocols. In B. Garbinato, H. Miranda, and L. Rodrigues, editors, *Middleware for Network Eccentric and Mobile Applications*, pages 97–120. Springer-Verlag, 2009.

[5] H. Attiya and J. Welch. *Distributed Computing: Fundamentals, Simulations, and Advanced Topics*. John Wiley and Sons, Inc., 2nd edition, 2004.

[6] B. Awerbuch, Y. Mansour, and N. Shavit. Polynomial end-to-end communication. In *Proc. of 30th Symp. on Foundations of Computer Science (FOCS)*, pages 358–363, 1989.

[7] B. Awerbuch, B. Patt-Shamir, D. Peleg, and M. E. Saks. Adapting to asynchronous dynamic networks. In *Proc. of the 24th Annual ACM Symposium on Theory of Computing (STOC)*, pages 557–570, 1992.

[8] B. Awerbuch and M. Sipser. Dynamic networks are as fast as static networks. In *Proc. of 29th Symp. on Foundations of Computer Science (FOCS)*, pages 206–220, 1988.

[9] R. Bar-Yehuda, O. Goldreich, and A. Itai. On the time complexity of broadcast in radio networks: An exponential gap between determinism and randomization. *Journal of Computer and System Sciences (JCSS)*, 45(1):104–126, 1992.

[10] H. Baumann, P. Crescenzi, and P. Fraigniaud. Parsimonious flooding in dynamic graphs. In *Proc. of 28th Symp. on Principles of Distributed Computing (PODC)*, pages 260–269, 2009.

[11] A. Clementi, C. Macci, A. Monti, F. Pasquale, and R. Silvestri. Flooding time in edge-markovian dynamic graphs. In *Proc. of 27th Symp. on Principles of Distributed Computing (PODC)*, pages 213–222, 2008.

[12] A. E. F. Clementi, A. Monti, F. Pasquale, and R. Silvestri. Broadcasting in dynamic radio networks. *J. Comput. Syst. Sci.*, 75(4):213–230, 2009.

[13] A. E. G. Clementi, A. Monti, and R. Silvestri. Distributed multi-broadcast in unknown radio networks. In *Proc. of 20th Symp. on Principles of Distributed Computing (PODC)*, pages 255–263, 2001.

[14] A. Cornejo, F. Kuhn, R. Ley-Wild, and N. A. Lynch. Keeping mobile robot swarms connected. In *Proc. of 23rd Conference on Distributed Computing (DISC)*, pages 496–511, 2009.

[15] S. Dolev. *Self-Stabilization*. MIT Press, 2000.

[16] M. Fischer and H. Jiang. Self-stabilizing leader election in networks of finite-state anonymous agents. In *Proc. of 10th Int. Conf. on Principles of Distributed Systems (OPODIS)*, pages 395–409, 2006.

[17] P. Flajolet and G. N. Martin. Probabilistic counting algorithms for data base applications. *Journal of Computer and System Sciences*, 31(2):182 – 209, 1985.

[18] E. Gafni and D. Bertsekas. Distributed algorithms for generating loop-free routes in networks with frequently changing topology. *IEEE Transactions on Communication*, 29(1):11–18, 1981.

[19] T. P. Hayes, J. Saia, and A. Trehan. The forgiving graph: A distributed data structure for low stretch under adversarial attack. In *Proc. of 28th Symp. on Principles of Distributed Computing (PODC)*, pages 121–130, 2009.

[20] S. M. Hedetniemi, S. T. Hedetniemi, and A. L. Liestman. A survey of gossiping and broadcasting in communication networks. *Networks*, 18:319–349, 1988.

[21] J. Hromkovič, R. Klasing, B. Monien, and R. Peine. Dissemination of information in interconnection networks (broadcasting & gossiping). *Combinatorial Network Theory*, pages 125–212, 1996.

[22] R. Ingram, P. Shields, J. E. Walter, and J. L. Welch. An asynchronous leader election algorithm for dynamic networks. In *Proc. of 23rd IEEE Int. Symp. on Parallel and Distributed Processing (IPDPS)*, pages 1–12, 2009.

[23] R. Karp, C. Schindelhauer, S. Shenker, and B. Vöcking. Randomized rumor spreading. In *Proc. of 41st Symp. on Foundations of Computer Science (FOCS)*, pages 565–574, 2000.

[24] D. Kempe, A. Dobra, and J. Gehrke. Gossip-based computation of aggregate information. In *Proc. of 44th Symp. on Foundations of Computer Science (FOCS)*, pages 482–491, 2003.

[25] D. Kempe and J. Kleinberg. Protocols and impossibility results for gossip-based communication mechanisms. In *Proc. of 43rd Symp. on Foundations of Computer Science (FOCS)*, pages 471–480, 2002.

[26] A. Korman. Improved compact routing schemes for dynamic trees. In *Proc. of 27th Symp. on Principles of Distributed Computing (PODC)*, pages 185–194, 2008.

[27] D. Kowalski and A. Pelc. Broadcasting in undirected ad hoc radio networks. In *Proc. of 22nd Symp. on Principles of Distributed Computing (PODC)*, pages 73–82, 2003.

[28] D. Krizanc, F. Luccio, and R. Raman. Compact routing schemes for dynamic ring networks. *Theory of Computing Systems*, 37:585–607, 2004.

[29] F. Kuhn, T. Locher, and R. Oshman. Gradient clock synchronization in dynamic networks. In *Proc. of 21st ACM Symp. on Parallelism in Algorithms and Architectures (SPAA)*, pages 270–279, 2009.

[30] F. Kuhn, N. A. Lynch, and C. C. Newport. The abstract MAC layer. In *Proc. of 23rd Conference on Distributed Computing (DISC)*, pages 48–62, 2009.

[31] F. Kuhn, S. Schmid, and R. Wattenhofer. A self-repairing peer-to-peer system resilient to dynamic adversarial churn. In *Proc. of 4th Int. Workshop on Peer-To-Peer Systems (IPTPS)*, 2005.

[32] L. Lamport. Time, clocks, and the ordering of events in a distributed system. *Commun. ACM*, 21(7):558–565, 1978.

[33] X. Li, M. J, and C. Plaxton. Active and Concurrent Topology Maintenance. In *Proc. of 18th Conference on Distributed Computing (DISC)*, 2004.

[34] N. A. Lynch. *Distributed Algorithms*. Morgan Kaufmann Publishers, 1996.

[35] N. Malpani, J. L. Welch, and N. Vaidya. Leader election algorithms for mobile ad hoc networks. In *DIALM '00: Proceedings of the 4th international workshop on Discrete algorithms and methods for mobile computing and communications*, pages 96–103, New York, NY, USA, 2000. ACM.

[36] D. Mosk-Aoyama and D. Shah. Computing separable functions via gossip. In *Proc. of 25th Symp. on Principles of Distributed Computing (PODC)*, pages 113–122, 2006.

[37] R. O'Dell and R. Wattenhofer. Information dissemination in highly dynamic graphs. In *Proc. of 9th Joint Workshop on Foundations of Mobile Computing (DIALM-POMC)*, pages 104–110, 2005.

[38] R. Olfati-Saber and R. M. Murray. Consensus problems in networks of agents with switching topology and time-delays. *IEEE Transactions on Automatic Control*, 49(9):1520–1533, 2004.

[39] W. Ren and R. W. Beard. Consensus of information under dynamically changing interaction topologies. In *Proc. of American Control Conference*, pages 4939–4944, 2004.

[40] D. M. Topkis. Concurrent broadcast for information dissemination. *IEEE Transactions on Software Engineering*, SE-11(10), 1985.

[41] J. E. Walter, J. L. Welch, and N. H. Vaidya. A mutual exclusion algorithm for ad hoc mobile networks. *Wireless Networks*, 7(6):585–600, 2001.