Distributed Computation in Dynamic Networks

Fabian Kuhn[1] fabian.kuhn@usi.ch
Nancy Lynch[2] lynch@csail.mit.edu
Rotem Oshman[2] rotem@csail.mit.edu

[1]Faculty of Informatics, University of Lugano, 6904 Lugano, Switzerland
[2]Computer Science and Artificial Intelligence Lab, MIT, Cambridge, MA 02139, USA

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 allows the study of the fundamental computation power of dynamic networks. In particular, it 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. Algorithms for the dynamic graph model must cope with these unceasing changes.

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 gossip problem, which requires the nodes to disseminate k pieces of information to all the nodes in the network. We show an $\Omega(n \log k)$ bound on gossip in 1-interval connected graphs against centralized algorithms, and an $\Omega(n + nk/T)$ bound on exchanging k pieces of information in T-interval connected graphs for a restricted class of randomized distributed algorithms.

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.

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.

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 our lower bounds cover randomized algorithms as well. 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 know nothing about the network, not even its size, 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., [12]), 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.

Contribution. In this paper we mainly study the following problems in the context of dynamic graphs.

- *Counting*, in which nodes must determine the size of the network.
- *k-gossip*, 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-gossip 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 gossip 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 gossip. When T is not known, we show that both problems can be solved in $O(\min\left\{n^2, n + n^2 \log n/T\right\})$ rounds.

We also give two lower bounds, both concerning token-forwarding algorithms for gossip. 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-gossip 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-gossip 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, as well as 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-gossip and counting even when computation is initiated by one node and the rest of the nodes are asleep (see Appendix E.5).

Related work. For static networks, information dissemination and basic network aggregation tasks have been extensively studied (see e.g. [5, 17, 30]). In particular, the k-gossip problem is analyzed in [36], 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. [8, 24]) or multiple messages [11, 27] 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 [18, 20, 22]; gossiping aggregation protocols that can be used to approximate the size of the system are described in [21, 32]. 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, 30]. 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 stabilization (e.g., [1, 6, 7, 37, 19]), 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 [13] for a comprehensive treatment of this topic. Another assumption, studied for example in [23, 25, 31], 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 [15], 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 widely studied. There is some work on handling nodes that join and leave continually in peer-to-peer overlay networks [16, 28, 29]. Most closely related to the problems studied here is [33], 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 [33] 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.

A variant of T-interval connectivity was used in [26], where two of the authors studied clock synchronization in *asynchronous* dynamic networks. In [26] 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.

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 Markovian processes [9, 10]. Similar edge-dynamic graphs have also been considered in control theory literature, e.g. [34, 35].

Finally, a somewhat related computational model is 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. 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)**.** A dynamic graph $G = (V, E)$ is said to be *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*-Gossip.** An instance of k-gossip is a pair (V, I), where $I : V \to \mathcal{P}(\mathcal{T})$ assigns a set of tokens from some token domain $\mathcal{T}$ to each node in the graph, and $|\bigcup_{u \in V} I(v)| = k$. An algorithm solves gossip 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 Gossip. A restricted class of k-gossip 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 gossip 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*-gossip in* 1*-interval connected graphs in* $n - 1$ *rounds, if nodes are not required to halt after they output the token.*[1]

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. At 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. □

If we have an upper bound on the size of the network, we can use Proposition 3.1 to compute simple functions which serve as building blocks for algorithms.

Proposition 3.2. *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.*

Proposition 3.1 guarantees that all nodes will have the min or max value after $n-1$ rounds; nodes need the upper bound N 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 [14, 32]. Please see Appendix E.6 for details.

The remainder of the paper focuses on counting and solving the gossip 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. The details are in Appendix E.1.2.

Proposition 3.3. *The counting and all-to-all gossip problems can be solved in* $O(n)$ *rounds in 1-interval connected graphs, using messages of size* $O(n \log n)$.

In the sequel we describe solutions which use only $O(\log n)$-bit messages.

[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.

4 Counting Through k-Committee Election in 1-Interval Connected Graphs

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

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 problem of 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.

In the case where $k < n$ 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 in the committee that still has $x = 1$ hears either a different committee ID or $\perp$. This node then sets $x \leftarrow 0$. 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 (synchronous start). 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 Protocol for k-Committee Election in 1-Interval Connected Graphs

To solve k-committee election, 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.

To handle the lack of a pre-elected leader, *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, any invitations they already issued remain in force.

Theorem 5.1. *The protocol sketched above solves the k-committee election problem in $O(k^2)$ rounds. When used in conjunction with the k-verification protocol it yields an $O(n^2)$-round counting protocol.* □

We remark that if $k \geq n$, the protocol also solves the gossip problem, if we use tokens wherever the protocol uses node UIDs. Each token is "singled out" for $k - 1 \geq n - 1$ rounds during which it is invited to join the leader's committee; the invitation is propagated by all nodes that receive it, and reaches all the nodes in the graph. Nodes simply have to record the tokens attached to invitations they hear. In particular, if node UIDs are used as tokens, nodes can collect all the UIDs they hear, and be guaranteed that if $k \geq n$ they have collected all UIDs in the network by the end of the protocol.

6 Counting and Gossip 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 T-Gossip 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 $k \geq n$, the procedure is guaranteed to provide each node with 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 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 $\mathrm{tdist}_i(u,t) \leq r \leq 2T$, 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 $r \geq \mathrm{tdist}_i(u,t)$, 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 $k \geq n$, then at the end of procedure `disseminate`, the set A_u of each node u contains the T smallest tokens.*

Proof Sketch. Let $N_i^d(t) := \{u \in V \mid \mathrm{tdist}_i(u,t) \leq d\}$. 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 $|N_i^T(t)| \geq \min\{n - |K_i(t)|, T\}$; that is, in each phase T 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. 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 General Gossip

To solve counting and gossip 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 or gossip protocol.*

6.3 Adapting to 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 $k \geq n$. 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 $k \geq n$, their error is one-sided: if their guess for T is too large they may falsely conclude that $k < n$ when in fact $k \geq n$, but they will never conclude that $k \geq n$ 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 $k \geq n$ 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 gossip 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 Lower Bounds on Gossip with Token-Forwarding Algorithms

Our algorithms for gossip 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 gossip with token-forwarding algorithms.

7.1 $\Omega(n \log k)$ Lower Bound for Centralized k-Gossip in 1-Interval Connected Graphs

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. 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 (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 7.1. *Any centralized algorithm for k-gossip in 1-interval connected graphs requires $\Omega(n \log k)$ rounds to complete in the worst case.*

Proof. We construct the communication graph for each round r in three stages. See Fig. 2 in the appendix for an illustration.

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 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 $\ell \geq 6$, 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, $Bottom$ according to the number of tokens they know: $Top := \{v_i \mid missing(v_i) \leq \ell/6\}$, $Bottom := \{v_i \mid missing(v_i) > \ell/6\}$.

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, and each bottom node will learn at most three new tokens. We begin by bounding the size of Top.

To that end, notice that $\sum_{u \in Top} missing(u) \geq \binom{|Top|}{2}$: 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. On the other hand, since each node in Top is missing at most $\ell/6$ tokens, it follows that $\sum_{u \in Top} missing(u) \leq |Top| \cdot (\ell/6)$. Putting the two facts together we obtain $|Top| \leq \ell/3 + 1$, and consequently also $|Bottom| = \ell - |Top| \geq 2\ell/3 - 1$.

Stage III: Connecting the nodes. The bottom nodes are relatively cheap to connect to, so we connect them in an arbitrary line. 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 using only the edges $P := \{\{u, v\} \mid u \in Top, v \in Bottom \text{ and } t_v \in A_u(r)\}$.

Since each top node is missing at most $\ell/6$ tokens, and each bottom node broadcasts a different value, for each top node there are at least $|Bottom| - \ell/6$ edges in P to choose from. But $|Top| \leq \ell/3 + 1 \leq |Bottom| - \ell/6$; thus, each top node can be connected to a different bottom node using P-edges.

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

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

7.2 $\Omega(n + nk/T)$ Lower Bound on k-Gossip with Knowledge-Based Algorithms

A token-forwarding randomized algorithm for k-gossip is *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). Knowledge-based algorithms can use the set of tokens currently known and 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 the UIDs of nodes from which a token was received. Nevertheless, the class of knowledge-based algorithms includes many natural strategies for solving the gossip problem, and it includes the algorithms in this paper.

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 7.2. *Any knowledge-based algorithm for k-gossip in T-interval connected graphs requires $\Omega(n + nk/T)$ rounds to succeed with probability $1/2$. Further, if $|\mathcal{U}| = \Omega(n^2 k/T)$, 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. (The bound holds for any assignment in which one node knows all tokens and a constant fraction of nodes is missing each token; for deterministic algorithms it holds when each node starts with one token.)

Let $r_1 = r_0 + (n-1)(k-1)/(4T) - 1$. Since u knows all the tokens, its behavior is determined: regardless of the dynamic graph we choose for the rounds between r_0 and r_1, the distribution of tokens broadcast by node u in rounds $r_0, \ldots, r_1$ is fixed. In particular, since there are less than $(n-1)(k-1)/(4T)$ rounds

between r_0 and r_1, the linearity of expectation and Markov's inequality show that there is some token t such that with probability at least $1/2$, node u broadcasts t less than $(n-1)/(2T)$ times between the two rounds.

In round r_0, token t is known only to node u; there are $n-1$ nodes that still must learn t. Informally, in the dynamic graph we construct node u will control the spread of t, such that after each time node u broadcasts t at most $2T$ nodes learn it. Thus, with probability at least $1/2$, by round r_1 there will still be a node that has not learned t, and the algorithm will not be done.

We construct the graph in phases. Initially we arrange the nodes in a line $u_0, v_1, \ldots, v_{n-1}$, with u_0 at one end of the line and the other nodes ordered arbitrarily (see Fig. 1(a) in the appendix). The nodes in the line can only learn t when u broadcasts it.

We leave the graph static until the first time node u broadcasts t. At this point we would like to remove v_1 from the line and place it in the clique, to prevent it from spreading t to other nodes in the line; however, we must respect T-interval connectivity. We proceed by first adding an edge from u to v_{n-1}, closing the line and forming a ring (see Fig. 1(b) in the appendix). A phase consists of waiting for T rounds with the graph static. Between phases we change the graph and remove $2T$ nodes that may have learned t from the ring. Next we describe how the graph changes between phases.

At the beginning of each phase the ring consists of a sequence $u, v_R, \ldots, v_L, u$. Initially $R = 1$ and $L = n-1$, and as the execution progresses we shrink the ring by moving nodes into the clique, increasing R and decreasing L. At the beginning of the phase, only nodes $u, v_R, \ldots, v_{R+T-1}$ can know token t (and in particular, in the first phase only v_R knows t), and only those nodes can spread it to other nodes. We call nodes $v_R, \ldots, v_{R+2T-1}$ *red nodes*, because these nodes may learn t during the phase even if u does not broadcast it. Nodes $v_L, \ldots, v_{L-T+1}$ are called *yellow nodes*, because they may learn t if node u broadcasts it during the phase. The rest of the nodes cannot learn t during the phase.

After the phase ends, we remove the red nodes from the ring and place them in the clique (see Fig. 1(c)). We restore the ring by adding an edge between u and v_{R+2T}. If u did not broadcast t during the phase that ended, then no node in the ring knows t, and we simply wait until u broadcasts t again. If u did broadcast t, we change the direction of the ring, setting $L \leftarrow R + 2T$ and $R \leftarrow L$. The nodes that were yellow now become red (see Fig. 1(d)), and we proceed as before. Each time u broadcasts t causes us to remove at most $2T$ nodes from the ring, but since with high probability t is broadcast less than $(n-1)/(2T)$ times, in round $r_1 = \Omega(nk/T)$ with probability at least $1/2$ the ring is not empty and the algorithm is not done. □

8 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 to the nodes. Nevertheless, we show that it is possible to efficiently compute any computable function, 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 type of 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 gossip and obtain an $\Omega(nk/T)$ general lower bound, and to determine whether counting is in fact as hard as gossip. 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.

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, 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.

[7] 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.

[8] 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.

[9] 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.

[10] 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.

[11] 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.

[12] 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.

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

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

[15] 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.

[16] 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.

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

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

[19] R. Ingram, P. Shields, J. E. Walter, and J. L. Welch. An asynchronous leader election algorithm for dynamic networks. In *IPDPS*, pages 1–12. IEEE, 2009.

[20] 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.

[21] 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.

[22] 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.

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

[24] 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.

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

[26] F. Kuhn, T. Locher, and R. Oshman. Gradient clock synchronization in dynamic networks. In F. M. auf der Heide and M. A. Bender, editors, *SPAA*, pages 270–279. ACM, 2009.

[27] 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.

[28] 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.

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

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

[31] 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.

[32] 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.

[33] 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.

[34] 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.

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

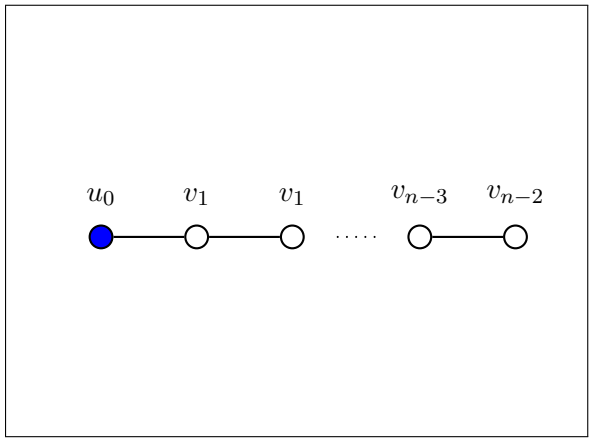
[36] D. M. Topkis. Concurrent broadcast for information dissemination. *IEEE Transactions on Software Engineering*, SE-11(10), 1985.

[37] 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.

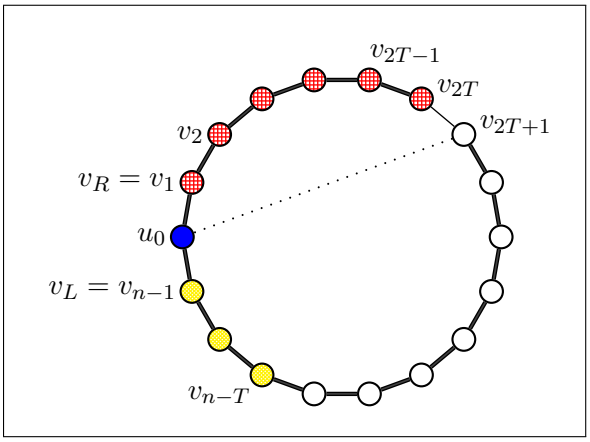
Appendix

In the following we begin with some figures to illustrate the lower bound proofs of Section 7. The remainder of the appendix contains a full version of all technical parts of the paper.

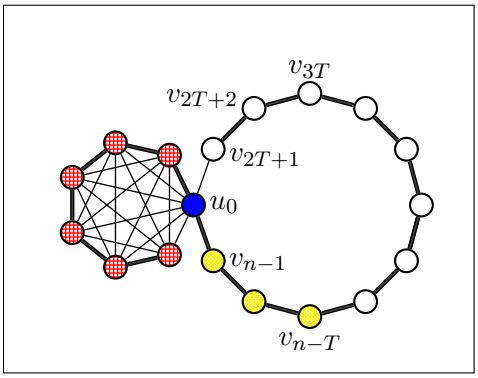
A Figures



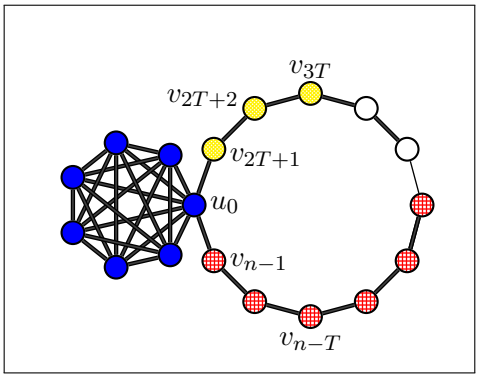
(a) The network at the beginning of the execution (only node u_0 knows t).



(b) The network at the beginning of the first phase: the line is closed to form a ring. The dotted line indicates the edge we will add at the end of the phase to re-close the ring after we remove the red nodes; double lines indicate stable edges, along which T-interval connectivity is preserved between phases.



(c) The network after the end of the first phase: the red nodes are removed from the ring and placed in the clique, and the ring is repaired by connecting u_0 to v_{2T+1}. Double lines indicate stable edges along which T-interval connectivity was preserved in the transition between the phases.



(d) If u_0 broadcast t at any point during the first phase, we begin a new phase. The nodes that were yellow in the first phase become red, and the "clean" nodes on u_0's other side become yellow. Double lines indicate edges that will be stable through the next two phases.

Figure 1: Illustrations for the proof of the $\Omega(n + nk/T)$ lower bound, $T = 3$

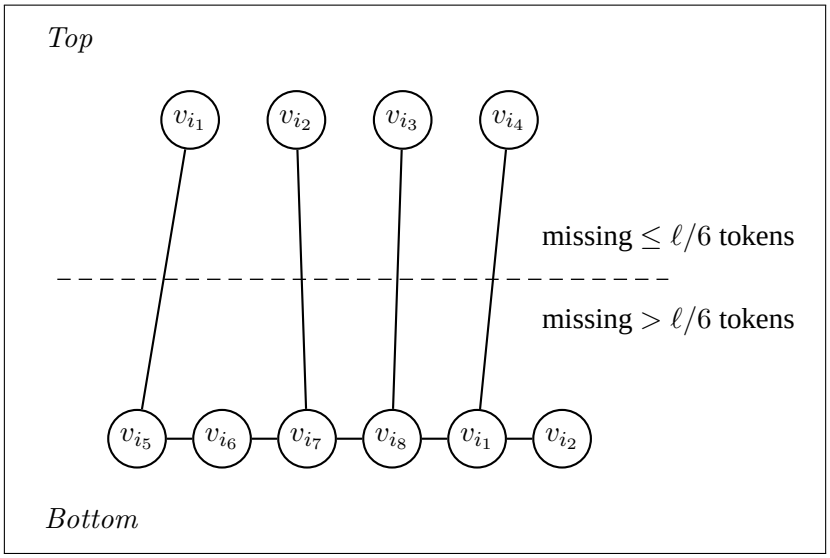


Figure 2: Illustration for the proof of the $\Omega(n \log k)$ lower bound

B Network Model

B.1 Dynamic Graphs

A synchronous dynamic network is modelled by a dynamic graph $G = (V, E)$, where V is a static set of nodes, and $E : \mathbb{N} \to V^{(2)}$ is a function mapping a round number $r \in \mathbb{N}$ to a set of undirected edges $E(r)$. Here $V^{(2)} := \{\{u, v\} \mid u, v \in V\}$ is the set of all possible undirected edges over V.

Definition B.1 (T-Interval Connectivity)**.** A dynamic graph $G = (V, E)$ is said to be *T-interval connected* for $T \in \mathbb{N}$ if for all $r \in \mathbb{N}$, the static graph $G_{r,T} := \left(V, \bigcap_{i=r}^{r+T-1} E(r)\right)$ is connected. If G is 1-interval connected we say that G is *always connected*.

Definition B.2 (∞-Interval Connectivity)**.** A dynamic graph $G = (V, E)$ is said to be *∞-interval connected* if there exists a connected static graph $G' = (V, E')$ such that for all $r \in \mathbb{N}$, $E' \subseteq E(r)$.

Note that even though in an ∞-interval connected graph there is some stable subgraph that persists throughout the execution, this subgraph is not known in advance to the nodes, and can be chosen by the adversary "in hindsight".

Although we are generally interested in the undirected case, it is also interesting to consider *directed dynamic graphs*, where the communication links are not necessarily symmetric. The T-interval connectivity assumption is then replaced by *T-interval strong connectivity*, which requires that $G_{r,T}$ be strongly connected (where $G_{r,T}$ is defined as before). In this very weak model, not only do nodes not know who will receive their message before they broadcast, they also do not know who received the message *after* it is broadcast. Interestingly, all of our algorithms for the undirected case work in the directed case as well.

The causal order for dynamic graphs is defined in the standard way.

Definition B.3 (Causal Order)**.** Given a dynamic graph $G = (V, E)$, we define an order $\to \subseteq (V \times \mathbb{N})^2$, where $(u, r) \to (v, r')$ iff $r' = r + 1$ and $\{u, v\} \in E(r)$. The *causal order* $\rightsquigarrow \subseteq (V \times \mathbb{N})^2$ is the reflexive and transitive closure of $\to$. We also write $u \rightsquigarrow (v, r)$ if there exists some $r' \leq r$ such that $(u, r') \rightsquigarrow (v, r)$.

Definition B.4 (Influence Sets)**.** We denote by $C_u(r \rightsquigarrow r') := \{v \in V \mid (v, r) \rightsquigarrow (u, r')\}$ the set of nodes whose state in round r causally influences node u in round r'. We also use the short-hand $C_u(r) := C_u(0 \rightsquigarrow r) = \{v \mid v \rightsquigarrow (u, r)\}$.

B.2 Communication and Adversary Model

Nodes communicate with each other using *anonymous broadcast*, with message sizes limited to $O(\log(n))$. At the beginning of round r, each node u decides what message to broadcast based on its internal state and private coin tosses; at the same time and independently, the adversary chooses a set $E(r)$ of edges for the round. For this choice the adversary can see the nodes' internal states at the beginning of the round, but not the results of their coin tosses or the message they have decided to broadcast. (Deterministic algorithms choose a message based only on the internal state, and this is equivalent to letting the adversary see the message before it chooses the edges.) The adversary then delivers to each node u all messages broadcast by nodes v such that $\{u, v\} \in E(r)$. Based on these messages, its previous internal state, and possibly more coin tosses, the node transitions to a new state, and the round ends. We call this anonymous broadcast because nodes do not know who will receive their message prior to broadcasting it.

B.3 Sleeping Nodes

Initially all nodes in the network are asleep; computation begins when a subset of nodes, chosen by the adversary, is woken up. Sleeping nodes remain in their initial state and do not broadcast any messages until they receive a message from some awake node or are woken up by the adversary. Then they wake up and

begin participating in the computation; however, since messages are delivered at the end of the round, a node that is awakened in round r sends its first message in round $r + 1$.

We refer to the special case where all nodes are woken up at once as *synchronous start*.

B.4 Initial Knowledge

Each node in the network starts execution of the protocol in an initial state which contains its own ID, its input, and possibly additional knowledge about the network. We generally assume one of the following.

- No knowledge: nodes know nothing about the network, and initially cannot distinguish it from any other network.
- Upper bound on size: nodes know some upper bound N on the size n of the network. The upper bound is assumed to be bounded by some function of the true size, e.g., $N = O(n)$.
- Exact size: nodes know the size n of the network.

B.5 Computation Model

We think of each node in the network as running a specialized Turing machine which takes the node's UID and input from its input tape at the beginning of the first round, and in subsequent rounds reads the messages delivered to the node from the input tape. In each round the machine produces a message to broadcast on an output tape. On a separate output tape, it eventually writes the final output of the node, and then enters a halting state.

The algorithms in this paper are written in pseudo-code. We use $x_u(r)$ to denote the value of node u's local variable x at the beginning of round r, and $x_u(0)$ to denote the input to node u.

C Problem Definitions

We assume that nodes have unique identifiers (UIDs) from some namespace $\mathcal{U}$. Let $\mathcal{D}$ be a problem domain. Further, let $A \mapsto B$ denote the set of all partial functions from A to B.

A *problem* over $\mathcal{D}$ is a relation $P \subseteq (\mathcal{U} \mapsto \mathcal{D})^2$, such that if $(I, O) \in P$ then $\mathrm{domain}(I)$ is finite and $\mathrm{domain}(I) = \mathrm{domain}(O)$. Each instance $I \in \mathcal{U} \mapsto \mathcal{D}$ induces a set $V = \mathrm{domain}(I)$ of nodes, and we say that an algorithm *solves* instance I if in any dynamic graph $G = (V, E)$, when each node $u \in V$ starts with $I(u)$ as its input, eventually each node outputs a value $O(u) \in \mathcal{D}$ such that $(I, O) \in P$.

We are interested in the following problems.

Counting. In this problem the nodes must determine the size of the network. Formally, the counting problem is given by

$$\mathsf{counting} := \{(V \times \{1\}, V \times \{n\}) \mid V \text{ is finite and } n = |V|\}.$$

***k*-Verification.** Closely related to counting, in the k-verification problem nodes are given an integer k and must determine whether or not $k \geq n$, eventually outputting a Boolean value. Formally,

$$\mathsf{k\text{-}verification} := \{(V \times \{k\}, V \times \{b\}) \mid b \in \{0, 1\} \text{ and } b = 1 \text{ iff } k \geq |V|\}.$$

***k*-Committee.** In this problem the nodes must form sets ("committees"), where each committee has a unique identifier that is known to all its members. Each node u outputs a value $committee_u$, and we require the following properties.

1. ("Safety") The size of each committee is at most k, that is, for all $x \in \{committee_u \mid u \in V\}$ we have $|\{u \in V \mid committee_u = x\}| \leq k$.
2. ("Liveness") If $k \geq n$ then all nodes in the graph join one committee, that is, for all $u, v \in V$ we have $committee_u = committee_v$.

***k*-Gossip.** The gossip problem is defined over a token domain $\mathcal{T}$. Each node receives in its input a set of tokens, and the goal is for all nodes to output all tokens. Formally,

$$\text{k-gossip} := \{(V \to \mathcal{P}(A), V \to A) \mid V \text{ is finite and } |A| = k\}.$$

We are particularly interested in the following variants of the problem.

- All-to-All gossip: instances I where $k = n$ for all $u \in V$ we have $|I(u)| = 1$.
- k-gossip with known k: in this variant nodes know k, i.e., they receive k as part of the input.

Leader Election. In weak leader election all nodes must eventually output a bit b, such that exactly one node outputs $b = 1$. In strong leader election, all nodes must output the same ID $u \in V$ of some node in the network.

D Relationships

A problem P_1 is *reducible* to P_2 if whenever all nodes start the computation in initial states that represent a solution to P_2, there is an algorithm that computes a solution to P_1 and requires linear time in the parameter to the problem (k).

D.1 *k*-Committee ≡ *k*-Verification

Claim D.1. *k-verification reduces to k-committee.*

Proof. Suppose we start from a global state that is a solution to k-committee, that is, each node u has a local variable $committee_u$ such that at most k nodes belong to the same committee, and if $k \geq n$ then all nodes belong to one committee. We can verify whether or not $k \geq n$ as follows. For k rounds, each node maintains a Boolean flag b, which is initially set to 1. In rounds where $b = 1$, the node broadcasts its committee ID, and when $b = 0$ the node broadcasts $\perp$. If a node receives a committee ID different from its own, or if it hears the special value $\perp$, it sets b to 0. At the end of the k rounds all nodes output b.

First consider the case where $k \geq n$. In this case all nodes have the same committee ID, and no node ever sets its b flag to 0. At the end of the protocol all nodes output 1, as required. Next, suppose that $k < n$, and let u be some node. There are at most $k - 1$ nodes in u's committee. In every round, there is an edge between some node in u's committee and some node in a different committee (because the communication graph is connected), and therefore at least one node in u's committee sets its b flag to 0. After at most k rounds no nodes remain, and in particular u itself must have $b_u = 0$. Thus, at the end of the protocol all nodes output 0. □

Claim D.2. *k-committee reduces to k-verification.*

Proof. Again, suppose the nodes are initially in a state that represents a solution to k-verification: they have a Boolean flag b which is set to 1 iff $k \geq n$. We solve k-committee as follows: if $b = 0$, then each node outputs its own ID as its committee ID. This is a valid solution because when $k < n$ the only requirement is that no committee have more than k nodes. If $b = 1$, then for k rounds all nodes broadcast the minimal ID they have heard so far, and at the end they output this ID as their committee ID. Since $b = 1$ indicates that $k \geq n$, after k rounds all nodes have heard the ID of the node with the minimal ID in the network, and they will all join the same committee, as required. □

D.2 Counting vs. *k*-Verification

Since we can solve k-verification in $O(k + k^2/T)$ time in T-interval connected graphs, we can find an upper bound on the size of the network by checking whether $k \geq n$ for values of k starting from 1 and doubling with every wrong guess. We know how to verify whether $k \geq n$ in $O(k + k^2/T)$ time, and hence the time complexity of the entire procedure is $O(n + n^2/T)$. Once we establish that $k \geq n$ for some value of k, to get

an actual count we can then go back and do a binary search over the range $k/2, \ldots, k$ (recall that $k/2 < n$, otherwise we would not have reached the current value of k).

In practice, we use a variant of k-committee where the ID of each committee is the set containing the IDs of all members of the committee. The k-verification layer returns this set as well, so that after reaching a value of $k \geq n$ at node u, we simply return the size of u's committee as the size of the network. Since $k \geq n$ implies that all nodes join the same committee, node u will output the correct count.

D.3 Hierarchy of Problems

There is a hardness hierarchy among the problems considered in this paper as well as some other natural problems.

1. Strong leader election / consensus (these are equivalent).
2. Decomposable functions such as Boolean AND / OR
3. Counting.
4. n-gossip (with unknown n).

The problems in every level are reducible to the ones in the next level, and we know that n-gossip can be solved in $O(n + n^2/T)$ time in T-interval connected graphs for $T \geq 2$, or $T \geq 1$ assuming synchronous start. Therefore all the problems can be solved in $O(n + n^2/T)$ time, even with no prior knowledge of the network, and even when the communication links are directed (assuming strong connectivity).

E Upper Bounds

In this section we give algorithms for some of the problems introduced in Section C, always with the goal of solving the counting problem. Our strategy is usually as follows:

1. Solve some variant of gossip.
2. Use (1) as a building block to solve k-committee,
3. Solving k-committee allows us to solve k-verification and therefore also counting (see Section D).

We initially focus on the case of synchronous start. The modifications necessary to deal with asynchronous start are described in Section E.5.

E.1 Always-Connected Graphs

E.1.1 Basic Information Dissemination

It is a basic fact that in 1-interval connected graphs, a single piece of information requires at most $n - 1$ rounds to reach all the nodes in the network, provided that it is forwarded by all nodes that receive it. Formally, let $D_u(r) := \{v \in V \mid u \rightsquigarrow (v, r)\}$ denote the set of nodes that u has "reached" by round r. If u knows a token and broadcasts it constantly, and all other nodes broadcast the token if they know it, then all the nodes in $D_u(r)$ know the token by round r.

Claim E.1. *For any node u and round $r \leq n - 1$ we have $|D_u(r)| \geq r + 1$.*

Proof. By induction on r. For $r = 0$ the claim is immediate. For the step, suppose that $|D_u(r)| \geq r+1$, and consider round $r+1 \leq n$. If $D_u(r) = V$ then the claim is trivial, because $D_u(r) \subseteq D_u(r+1)$. Thus, suppose that $D_u(r) \neq V$. Since $G(r)$ is connected, there is some edge $\{x, y\}$ in the cut $(D_u(r), V \setminus D_u(r))$. From the definition of the causal order we have $x, y \in D_u(r+1)$, and therefore $|D_u(r+1)| \geq |D_u(r)|+1 \geq r+2$. □

Note that we can employ this property even when there is more than one token in the network, provided that tokens form a totally-ordered set and nodes forward the smallest (or biggest) token they know. It is then guaranteed that the smallest (resp. biggest) token in the network will be known by all nodes after at most $n - 1$ rounds. Note, however, that in this case nodes do not necessarily *know* when they know the smallest or biggest token.

E.1.2 Counting in linear time with $\Omega(n \log n)$-bit messages

We begin by describing a linear-time counting/n-gossip protocol which uses messages of size $\Omega(n \log n)$. The protocol is extremely simple, but it demonstrates some of the ideas used in some of our later algorithms, where we eliminate the large messages using a stability assumption (T-interval connectivity) which allows nodes to communicate with at least one of their neighbors for at least T rounds.

In the simple protocol, all nodes maintain a set A containing all the IDs (or tokens) they have collected so far. In every round, each node broadcasts A and adds any IDs it receives. Nodes terminate when they first reach a round r in which $|A| < r$.

$A \leftarrow \{self\}$
for $r = 1, 2, \ldots$ **do**
  broadcast A
  receive $B_1, \ldots, B_s$ from neighbors
  $A \leftarrow A \cup B_1 \cup \ldots \cup B_s$
  if $|A| < r$ **then** terminate and output $|A|$

Algorithm 2: Counting in linear time using large messages

Claim E.2. *For any node u and rounds $r \leq r' \leq n$ we have $|C_u(r \rightsquigarrow r')| \geq r' - r$.*

Proof. By induction on $r' - r$. For $r' - r = 0$ the claim is immediate.

Suppose that for all nodes u and rounds r, r' such that $r' \leq n$ and $r' - r = i$ we have $|C_u(r \rightsquigarrow r')| \geq i$. Let $r, r' \leq n$ be two rounds such that $r' - r = i + 1$.

If $|C_u((r+1) \rightsquigarrow r)| = n$ then we are done, because $r' - r \leq r' \leq n$. Thus, assume that $C_u((r+1) \rightsquigarrow r) \neq V$. Since the communication graph in round r is connected, there is some edge $\{w, w'\} \in E(r)$ such that $w \notin C_u((r+1) \rightsquigarrow r)$ and $w' \in C_u((r+1) \rightsquigarrow r)$. We have $(w, r) \rightarrow (w', r+1) \rightsquigarrow (u, r')$, and consequently $(w, r) \rightsquigarrow (u, r')$ and $w \in C_u(r \rightsquigarrow r')$. Also, from the induction hypothesis, $|C_u((r+1) \rightsquigarrow r)| \geq i$. Together we obtain $|C_u(r \rightsquigarrow r')| \geq |C_u((r+1) \rightsquigarrow r)| + 1 \geq i + 1$, as desired. □

Claim E.3. *For any node u and round $r \leq n$ we have $|A_u(r)| \geq r$.*

Proof. It is easily shown that for all $v \in C_u(r)$ we have $v \in A_u(r)$. From the previous claim we have $|C_u(r)| \geq r$ for all $r \leq n$, and the claim follows. □

The correctness of the protocol follows from Claim E.3: suppose that for some round r and node u we have $|A_u(r)| < r$. From Claim E.3, then, $r > n$. Applying the claim again, we see that $|A_u(n)| \geq n$, and since $A_u(r) \subseteq V$ for all r, we obtain $A_u(r) = V$. This shows that nodes compute the correct count. For termination we observe that the size of A_u never exceeds n, so all nodes terminate no later than round $n+1$.

E.1.3 k-committee with $O(\log n)$-bit messages

We can solve k-committee in $O(k^2)$ rounds as follows. Each node u stores a local variable $leader_u$ in addition to $committee_u$. A node that has not yet joined a committee is called *active*, and a node that has joined a committee is *inactive*. Once nodes have joined a committee they do not change their choice.

Initially all nodes consider themselves leaders, but throughout the protocol, any node that hears an ID smaller than its own adopts that ID as its leader. The protocol proceeds in k cycles, each consisting of two phases, *polling* and *selection*.

1. Polling phase: for $k - 1$ rounds, all nodes propagate the ID of the smallest active node of which they are aware.

2. Selection phase: in this phase, each node that considers itself a leader selects the smallest ID it heard in the previous phase and invites that node to join its committee. An invitation is represented as a pair (x, y), where x is the ID of the leader that issued the invitation, and y is the ID of the invited node. All nodes propagate the smallest invitation of which they are aware for $k-1$ (invitations are sorted in lexicographic order, so that invitations issued by the smallest node in the network will win out over other invitations. It turns out, though, that this is not necessary for correctness; it is sufficient for each node to forward an arbitrary invitation from among those it received).
   At the end of the selection phase, a node that receives an invitation to join its leader's committee does so and becomes inactive. (Invitations issued by nodes that are not the current leader can be accepted or ignored; this, again, does not affect correctness.)

At the end of the k cycles, any node u that has not been invited to join a committee outputs $committee_u = u$.

```
leader ← self
committee ← ⊥
for i = 0, ..., k do
    // Polling phase
    if committee = ⊥ then
        min_active ← self ;            // The node nominates itself for selection
    else
        min_active ← ⊥
    for j = 0, ..., k − 1 do
        broadcast min_active
        receive x_1, ..., x_s from neighbors
        min_active ← min{min_active, x_1, ..., x_s}
    // Update leader
    leader ← min{leader, min_active}
    // Selection phase
    if leader = self then
        // Leaders invite the smallest ID they heard
        invitation ← (self, min_active)
    else
        // Non-leaders do not invite anybody
        invitation ← ⊥
    for j = 0, ..., k − 1 do
        broadcast invitation
        receive y_1, ..., y_s from neighbors
        invitation ← min{invitation, y_1, ..., y_s} ;      // (in lexicographic order)
    // Join the leader's committee, if invited
    if invitation = (leader, self) then
        committee = leader

if committee = ⊥ then
    committee ← self
```

Algorithm 3: k-committee in always-connected graphs

Claim E.4. *The protocol solves the k-committee problem.*

Proof. We show that after the protocol ends, the values of the local $committee_u$ variables constitute a valid

solution to k-committee.

1. In each cycle, each node invites at most one node to join its committee. After k cycles at most k nodes have joined any committee. Note that the first node invited by a leader u to join u's committee is always u itself. Thus, if after k cycles node u has not been invited to join a committee, it follows that u did not invite any other node to join its committee; when it forms its own committee in the last line of the algorithm, the committee's size is 1.
2. Suppose that $k \geq n$, and let u be the node with the smallest ID in the network. Following the polling phase of the first cycle, all nodes v have $leader_v = u$ for the remainder of the protocol. Thus, throughout the execution, only node u issues invitations, and all nodes propagate u's invitations. Since $k \geq n$ rounds are sufficient for u to hear the ID of the minimal active node in the network, in every cycle node u successfully identifies this node and invites it to join u's committee. After k cycles, all nodes will have joined.

□

Remark. The protocol can be modified easily to solve n-gossip if $k \geq n$. Let t_u be the token node u received in its input (or $\perp$ if node u did not receive a token). Nodes attach their tokens to their IDs, and send pairs of the form (u, t_u) instead of just u. Likewise, invitations now contain the token of the invited node, and have the structure $(leader, (u, t_u))$. The min operation disregards the token and applies only to the ID. At the end of each selection phase, nodes extract the token of the invited node, and add it to their collection. By the end of the protocol every node has been invited to join the committee, and thus all nodes have seen all tokens.

E.2 ∞-interval Connected Graphs

We can count in linear time in ∞-interval connected graphs using the following algorithm: each node maintains two sets of IDs, A and S. A is the set of all IDs known to the node, and S is the set of IDs the node has already broadcast. Initially A contains only the node's ID and S is empty. In every round, each node broadcasts $\min(A \setminus S)$ and adds this value to S. (If $A = S$, the node broadcasts nothing.) Then it adds all the IDs it receives from its neighbors to A.

While executing this protocol, nodes keep track of the current round number (starting from zero). When a node reaches a round r in which $|A| < \lfloor r/2 \rfloor$, it terminates and outputs $|A|$ as the count.

```
S ← ∅
A ← {self}
for r = 0, ... 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}
    if |A| < ⌊r/2⌋ then terminate and output |A|
return A
```

Algorithm 4: Counting in ∞-interval connected graphs

E.2.1 Analysis

Let $\mathrm{dist}(u, v)$ denote the shortest-path distance between u and v in the stable subgraph G', and let $N^d(u)$ denote the d-neighborhood of u in G', that is, $N^d(u) = \{v \in V \mid \mathrm{dist}(u, v) \leq d\}$. We use $A_x(r)$ and $S_x(r)$ to denote the values of local variables A and S at node $x \in V$ in the beginning of round r. Note the following properties:

1. $S_x(r+1) \subseteq A_x(r) \subseteq A_x(r+1)$ for all x and r.

2. If u and v are neighbors in G', then $S_u(r) \subseteq A_v(r)$ for all r, because every value sent by u is received by v and added to A_v.
3. S and A are monotonic, that is, for all x and r we have $S_x(r) \subseteq S_x(r+1)$ and $A_x(r) \subseteq A_x(r+1)$.

Claim E.5. *For every two nodes $x, u \in V$ and round r such that $r \geq \mathrm{dist}(u,x)$, either $x \in S_u(r+1)$ or $|S_u(r+1)| \geq r - \mathrm{dist}(u,x)$.*

Proof. By induction on r. For $r = 0$ the claim is immediate.

Suppose the claim holds for round $r-1$, and consider round r. Let x, u be nodes such that $r \geq \mathrm{dist}(u,x)$; we must show that either $x \in S_u(r+1)$ or $|S_u(r+1)| \geq r - \mathrm{dist}(u,x)$.

If $x = u$, then the claim holds: u is broadcast in the first round, and thereafter we have $u \in S_u(r)$ for all $r \geq 1$.

Otherwise, let v be a neighbor of u along the shortest path from u to x in G'; that is, v is a neighbor of u such that $\mathrm{dist}(v,x) = \mathrm{dist}(u,x) - 1$. Since $r \geq \mathrm{dist}(u,x) = \mathrm{dist}(v,x) + 1$ we have $r - 1 \geq \mathrm{dist}(v,x)$.

From the induction hypothesis on v and x in round $r-1$, either $x \in S_v(r)$ or $|S_v(r)| \geq r-1-\mathrm{dist}(v,x) = r - \mathrm{dist}(u,x)$. Applying property 2 above, this implies the following.

($\star$) Either $x \in A_u(r)$ or $|A_u(r)| \geq r - \mathrm{dist}(u,x)$.

If $x \in S_u(r)$ or $|S_u(r)| \geq r - \mathrm{dist}(u,x)$ then we are done, because $S_u(r) \subseteq S_u(r+1)$. Suppose then that $x \notin S_u(r)$ and $|S_u(r)| < r - \mathrm{dist}(u,x)$. It is sufficient to prove that $A_u(r) \neq S_u(r)$: this shows that in round r node u broadcasts $\min(A_u(r) \setminus S_u(r))$ and adds it to S_u, yielding $|S_u(r+1)| \geq |S_u(r)| + 1 \geq r - \mathrm{dist}(u,x)$ and proving the claim.

We show this using ($\star$). If $x \in A_u(r)$, then $A_u(r) \neq S_u(r)$, because we assumed that $x \notin S_u(r)$. Otherwise ($\star$) states that $|A_u(r)| \geq r - \mathrm{dist}(u,x)$, and since we assumed that $|S_u(r)| < r - \mathrm{dist}(u,x)$, this again shows that $A_u(r) \neq S_u(r)$. □

Claim E.6. *If $r \leq n$, then for all nodes u we have $|A_u(2r)| \geq r$.*

Proof. Let $u \in V$. For any node $x \in N^r(u)$, Claim E.5 shows that either $x \in S_u(2r+1)$ or $|S_u(2r+1)| \geq 2r - \mathrm{dist}(u,x) \geq r$. Thus, either $|S_u(2r+1)| \geq r$ or $N^r(u) \subseteq S_u(2r+1)$. Since $r \leq n$ and G' is connected we have $N^r(u) \geq r$, and therefore in both cases we have $|A_u(2r)| \geq |S_u(2r+1)| \geq r$. □

Claim E.7. *The algorithm terminates in linear time and outputs the correct count at all nodes.*

Proof. Termination is straightforward: the set A only contains IDs of nodes that exist in the network, so its size cannot exceed n. All nodes terminate no later than round $2n+2$.

Correctness follows from Claim E.6. Suppose that in round r node u has $|A_u(r)| < \lfloor r/2 \rfloor$, and let $r' = \lfloor r/2 \rfloor$. We must show that $A_u(r) = V$.

From Claim E.6, if $r' \leq n$ then $|A_u(2r')| \geq r'$. By definition of r' we have $r \geq 2r'$ and hence from Property 3 we obtain $|A_u(r)| \geq r'$, which is not the case. Thus, $r' > n$ and $r > 2n$. Applying the same reasoning as in Claim E.6 to round n, we see that either $|S_u(2n+1)| > n$ or $N^n(u) \subseteq S_u(2n+1)$. Since the first cannot occur it must be the case that $V = N^n(u) \subseteq S_u(2n+1) \subseteq A_u(r)$, and we are done. □

E.3 Finite-Interval Connected Graphs

Next we generalize the protocol above, in order to solve k-committee in $2T$-interval connected graphs. The general protocol requires $O(n + n^2/T)$ rounds (and assumes that T is known in advance). The idea is the same as for always-connected graphs, except that instead of selecting one node at a time to join its committee, each leader selects a batch of T nodes and disseminates their IDs throughout the network. We generalize and refine Claim E.5 for the case where there are initially up to n tokens, but only the smallest T tokens need to be disseminated.

E.3.1 T-gossip in $2T$-interval connected graphs

The "pipelining effect" we used in the ∞-interval connected case allows us to disseminate T tokens in $2n$ rounds, given that the graph is $2T$-interval connected. The idea is to use a similar protocol to the ∞-interval connected case, except that the protocol is "restarted" every $2T$ rounds: all nodes empty the set S (but not A), which causes them to re-send the tokens they already sent, starting from the smallest and working upwards. The T smallest tokens will thus be propagated through the network, and larger tokens will "die out" as they are not re-sent.

This is captured formally by the following protocol. The tokens are now assumed to come from a well-ordered set $(P, <)$. The input at each node u is an initial set $A_u \subseteq P$ of tokens. In addition, it is assumed that all nodes have a common guess k for the size of the network. The protocol guarantees that the T smallest tokens in the network are disseminated to all nodes, provided that the graph is $2T$-interval connected and that $k \geq n$.

```
S ← ∅
for i = 0, ..., ⌈k/T⌉ − 1 do
    for r = 0, ..., 2T 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
```

Function `disseminate`(A, T, k)

We refer to each iteration of the inner loop as a *phase*. Since a phase lasts $2T$ rounds and the graph is $2T$-interval connected, there is some connected subgraph that exists throughout the phase. Let G'_i be a connected subgraph that exists throughout phase i, for $i = 0, \ldots, \lceil k/T \rceil - 1$. We use $\mathrm{dist}_i(u, v)$ to denote the distance between nodes $u, v \in V$ in G'_i.

Let $K_t(r)$ denote the set of nodes that know token t by the beginning of round r, that is, $K_t(r) = \{u \in V \mid t \in A_u(r)\}$. In addition, let I be the set of T smallest tokens in $\bigcup_{u \in V} A_u(0)$. Our goal is to show that when the protocol terminates we have $K_t(r) = V$ for all $t \in I$.

For a node $u \in V$, a token $t \in P$, and a phase i, we define $\mathrm{tdist}_i(u, t)$ to be the distance of u from the nearest node in G'_i that knows t at the beginning of phase i:

$$\mathrm{tdist}(u, t) := \min \{\mathrm{dist}_i(u, v) \mid v \in K_t(2T \cdot i)\}.$$

Here and in the sequel, we use the convention that $\min \emptyset := \infty$. For convenience, we use $S^i_u(r) := S_u(2T \cdot i + r)$ to denote the value of S_u in round r of phase i. Similarly we denote $A^i_u(r) := A_u(2T \cdot i + r)$ and $K^i_t(r) := K_t(2T \cdot i + r)$.

The following claim characterizes the spread of each token in each phase. It is a generalization of Claim E.5, and the proof is similar.

Claim E.8. *For any node $u \in V$, token $t \in \bigcup_{u \in V} A_u(0)$ and round $r \in \{0, \ldots, 2T - 1\}$ such that $r \geq \mathrm{tdist}_i(u, t)$, either $t \in S^i_u(r + 1)$ or $S^i_u(r + 1)$ includes at least $(r - \mathrm{tdist}_i(u, t))$ tokens that are smaller than t.*

Proof. By induction on r. For $r = 0$ the claim is immediate.

Suppose the claim holds for round $r-1$ of phase i, and consider round $r \geq \mathrm{tdist}_i(u,t)$. If $r = \mathrm{tdist}_i(u,t)$, then $r - \mathrm{tdist}_i(u,t) = 0$ and the claim holds trivially. Thus, suppose that $r > \mathrm{tdist}_i(u,t)$. Hence, $r - 1 \geq \mathrm{tdist}_i(u,t)$, and the induction hypothesis applies: either $t \in S_u^i(r)$ or $S_u^i(r)$ includes at least $(r - 1 - \mathrm{tdist}_i(u,t))$ tokens that are smaller than t. In the first case we are done, since $S_u^i(r) \subseteq S_u^i(r+1)$; thus, assume that $t \notin S_u^i(r)$, and $S_u^i(r)$ includes at least $(r - 1 - \mathrm{tdist}_i(u,t))$ tokens smaller than t. However, if $S_u^i(r)$ includes at least $(r - \mathrm{tdist}_i(u,t))$ tokens smaller than t, then so does $S_u^i(r+1)$, and the claim is again satisfied; thus we assume that $S_u^i(r)$ includes *exactly* $(r - 1 - \mathrm{tdist}_i(u,t))$ tokens smaller than t.

It is sufficient to prove that $\min\left(A_u^i(r) \setminus S_u^i(r)\right) \leq t$: if this holds, then in round r node u broadcasts $\min\left(A_u^i(r) \setminus S_u^i(r)\right)$, which is either t or a token smaller than t; thus, either $t \in S_u^i(r+1)$ or $S_u^i(r+1)$ includes at least $(r - \mathrm{tdist}_i(u,t))$ tokens smaller than t, and the claim holds.

First we handle the case where $\mathrm{tdist}_i(u,t) = 0$. In this case, $t \in A_u^i(0) \subseteq A_u^i(r)$. Since we assumed that $t \notin S_u^i(r)$ we have $t \in A_u^i(r) \setminus S_u^i(r)$, which implies that $\min\left(A_u^i(r) \setminus S_u^i(r)\right) \leq t$.

Next suppose that $\mathrm{tdist}_i(u,t) > 0$. Let $x \in K_t^i(0)$ be a node such that $\mathrm{dist}_i(u,x) = \mathrm{tdist}(u,t)$ (such a node must exist from the definition of $\mathrm{tdist}_i(u,t)$), and let v be a neighbor of u along the path from u to x in G_i', such that $\mathrm{dist}_i(v,x) = \mathrm{dist}_i(u,x) - 1 < r$. From the induction hypothesis, either $t \in S_v^i(r)$ or $S_v^i(r)$ includes at least $(r - 1 - \mathrm{tdist}_i(v,t)) = (r - \mathrm{tdist}_i(u,t))$ tokens that are smaller than t. Since the edge between u and v exists throughout phase i, node u receives everything v sends in phase i, and hence $S_v^i(r) \subseteq A_u^i(r)$. Finally, because we assumed that $S_u^i(r)$ contains exactly $(r - 1 - \mathrm{tdist}_i(u,t))$ tokens smaller than t, and does not include t itself, we have $\min\left(A_u^i(r) \setminus S_u^i(r)\right) \leq t$, as desired. □

Claim E.9. *For each of the T smallest tokens $t \in I$ and phases i, we have $|K_t^i(0)| \geq \min\{n, T \cdot i\}$.*

Proof. The proof is by induction on i. For $i = 0$ the claim is immediate. For the induction step, suppose that $|K_t^i(0)| \geq \min\{n, T \cdot i\}$, and consider phase $i+1$.

Let $N(t)$ denote the T-neighborhood of $K_t^i(0)$, that is, $N(t) := \{u \in V \mid \mathrm{tdist}_i(u,t) \leq T\}$. From Claim E.8 applied to round $2T$ of phase i, for all $u \in N(t)$, either $t \in S_u^i(r+1)$ or $S_u^i(r+1)$ includes at least $2T - T = T$ tokens smaller than t. Since t is one of the T smallest tokens in the network, this latter case is impossible. Thus, every node $u \in N(t)$ has $t \in S_u^i(2T+1) \subseteq A_u^i(2T+1)$, which implies that $N(t) \subseteq K_t^{i+1}(0)$. In addition, $K_t^i(0) \subseteq K_t^{i+1}(0)$, because nodes never forget tokens they have learned.

Since G_i' is connected, $|N(t) \setminus K_t^i(0)| \geq T$. Combining with the induction hypothesis we obtain $|N(t) \cup K_t^i(0)| \geq \min\{n, T \cdot (i+1)\}$, and the claim follows. □

Procedure `disseminate` terminates at the end of phase $\lceil k/T \rceil - 1$, or, equivalently, at the beginning of phase $\lceil k/T \rceil$. By this time, if the guess for the size of the network was correct, all nodes have learned the T smallest tokens.

Corollary E.10. *If $k \geq n$, then $K_t^{\lceil k/T \rceil}(0) = V$ for each of the T smallest tokens $t \in I$.*

Proof. The claim follows from Claim E.9, because $T \cdot \lceil k/T \rceil \geq k \geq n$. □

E.3.2 k-committee in $2T$-interval connected graphs

We can solve the k-committee problem in $O(k + k^2/T)$ rounds using Algorithm 6. The idea is similar to Algorithm 3, except that leaders invite T nodes to join their committee in every cycle instead of just one node. Each node begins the protocol with a unique ID which is stored in the local variable *self*.

```
leader ← self
committee ← ⊥
for i = 0, ..., ⌈k/T⌉ − 1 do
    if committee = ⊥ then
        A ← {self} ;                    // The node nominates itself for selection
    else
        A ← ∅
    tokens ← disseminate(A, T, k)
    leader ← min ({leader} ∪ tokens)
    if leader = self then
        // Leaders invite the T smallest IDs they collected
        // (or less in the final cycle, so that the total does not exceed k)
        if i < ⌈k/T⌉ − 1 then
            A ← smallest-T(tokens)
        else
            m ← k − (⌈k/T⌉ − 1) · T
            A ← smallest-T(tokens)
    else
        // Non-leaders do not invite anybody
        A ← ∅
    tokens ← disseminate({self} × A, T, k)
    // Join the leader's committee, if invited
    if (leader, self) ∈ tokens then
        committee = leader
if committee = ⊥ then
    committee ← self
```

Algorithm 6: k-committee in $2T$-interval connected graphs

Claim E.11. *The protocol above solves k-committee in $O(k + k^2/T)$ rounds.*

E.3.3 Counting in Graphs with Unknown Finite-Interval Connectivity

The protocol 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 if $k \geq n$. However, even when the graph is not $2T$-interval connected, no committee 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 k-committee protocol above to solve k-verification, their error is one-sided: if their guess for T is too large they may falsely conclude that $k < n$ when in fact $k \geq n$, but they will never conclude that $k \geq n$ 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* using the following scheme. I assume the version of k-verification that returns the set V of all nodes if $k \geq n$, or the special value $\perp$ if $k < n$.

for $i = 1, 2, 4, 8, \ldots$ **do**
  for $k = 1, 2, 4, \ldots, i$ **do**
    if k*-verification assuming* $\lfloor k^2/i \rfloor$*-interval connectivity returns* $V \neq \perp$ **then**
      return $|V|$

Algorithm 7: Counting in $O(n \log n + n^2 \log(n)/T)$ in T-interval connected graphs where T is unknown

The time required for k-verification assuming $\lfloor k^2/i \rfloor$-interval connectivity is $O(k^2/\lfloor k^2/i \rfloor) = O(i)$ for all k, and thus the total time complexity of the i-th iteration of the outer loop is $O(i \log i)$.

If the communication graph is T-interval connected, the algorithm terminates the first time we reach values of i and k such that $k \geq n$ and $\lfloor k^2/i \rfloor \leq T$. Let N be the smallest power of 2 that is no smaller than n; clearly $N < 2n$. Let us show that the algorithm terminates when we reach $i = \max\left\{N, \lceil N^2/T \rceil\right\}$.

First consider the case where $\max\left\{N, \lceil N^2/T \rceil\right\} = N$, and hence $T \geq N$. When we reach the last iteration of the inner loop, where $k = i = N$, we try to solve N-verification assuming N-interval connectivity. This must succeed, and the algorithm terminates.

Next, suppose that $\lceil N^2/T \rceil > N$. Consider the iteration of the inner loop in which $k = N$. In this iteration, we try to solve N-verification assuming $\lfloor N^2/\lceil N^2/T \rceil \rfloor$-interval connectivity. Since $\lfloor N^2/\lceil N^2/T \rceil \rfloor \leq T$, this again must succeed, and the algorithm terminates.

The time complexity of the algorithm is dominated by the last iteration of the outer loop, which requires $O(i \log i) = O(n \log n + n^2 \log(n)/T)$ rounds.

The asymptotic time complexity of this algorithm only improves upon the original $O(n^2)$ algorithm (which assumes only 1-interval connectivity) when $T = \omega(\log n)$. However, it is possible to execute both algorithms in parallel, either by doubling the message sizes or by interleaving the steps, so that the original algorithm is executed in even rounds and Alg. 7 is executed in odd rounds. This will lead to a time complexity of $O(\min\left\{n^2, n \log n + n^2 \log(n)/T\right\})$, because we terminate when either algorithm returns a count.

E.4 Exploiting Expansion Properties of the Communication Graph

Naturally, if the communication graph is always a good expander, the algorithms presented here can be made to terminate faster. We consider two examples of graphs with good expansion. As before, when the expansion is not known in advance we can guess it, paying a $\log n$ factor.

E.4.1 f-Connected Graphs

Definition E.1. A static graph G is *f-connected* for $f \in \mathbb{N}$ if the removal of any set of at most $f - 1$ nodes from G does not disconnect it.

Definition E.2 (T-interval f-connectivity)**.** A dynamic graph $G = (V, E)$ is said to be *T-interval f-connected* for $T, f \in \mathbb{N}$ if for all $r \in \mathbb{N}$, the static graph $G_{r,T} := \left(V, \bigcap_{i=r}^{r+T-1} E(r)\right)$ is f-connected.

Definition E.3 (Neighborhoods)**.** Given a static graph $G = (V, E)$ and a set $S \subseteq V$ of nodes, the *neighborhood* of S in G is the set $\Gamma_G(S) = S \cup \{v \in V \mid \exists u \in S : \{u, v\} \in E\}$. The *$d$-neighborhood* of S is defined inductively, with $\Gamma^0_G(S) = S$ and $\Gamma^d_G(S) = \Gamma_G(\Gamma^{d-1}_G(S))$ for $d > 0$. We omit the subscript G when it is obvious from the context.

In f-connected graphs the propagation speed is multiplied by f, because every neighborhood is connected to at least f external nodes (if there are fewer than f remaining nodes, it is connected to all of them). This is shown by the following lemma.

Lemma E.12 (Neighborhood Growth)**.** *If $G = (V, E)$ is a static f-connected graph, then for any non-empty set $S \subseteq V$ and integer $d \geq 0$, we have $|\Gamma^d(S)| \geq \min\{|V|, |S| + fd\}$.*

Proof. By induction on d. For $d = 0$ the claim is immediate. For the step, suppose that $|\Gamma^d(S)| \geq \min\{|V|, |S| + fd\}$. Suppose further that $\Gamma^{d+1}(S) \neq V$, otherwise the claim is immediate. This also implies that $\Gamma^d(S) \neq V$, because $\Gamma^d(S) \subseteq \Gamma^{d+1}(S)$. Thus the induction hypothesis states that $|\Gamma^d(S)| \geq |S| + fd$.

Let $\Gamma := \Gamma^{d+1}(S) \setminus \Gamma^d(S)$ denote the "new" nodes in the $(d+1)$-neighborhood of S. It is sufficient to show that $|\Gamma| \geq f$, because then $|\Gamma^{d+1}(S)| = |\Gamma^d(S)| + |\Gamma| \geq |S| + f(d+1)$, and we are done.

Suppose by way of contradiction that $|\Gamma| < f$, and let $G' = (V', E')$ be the subgraph obtained from G by removing the nodes in Γ. Because G is f-connected and $|\Gamma| < f$, the subgraph G' is connected. Consider the cut $(\Gamma^d(S), V' \setminus \Gamma^d(S))$ in G'. Because $S \neq \emptyset$ and $S \subseteq \Gamma^d(S)$, we have $\Gamma^d(S) \neq \emptyset$, and because $\Gamma^d(S) \subseteq \Gamma^{d+1}(S)$ and $\Gamma^{d+1}(S) \neq V$, we also have $V' \setminus \Gamma^d(S) \neq \emptyset$. However, the cut is empty: if there were some edge $\{u, v\} \in E$ such that $u \in \Gamma^d(S)$ and $v \in V' \setminus \Gamma^d(S)$, then by definition of $\Gamma^{d+1}(S)$ we would have $v \in \Gamma^{d+1}(S)$. This in turn would imply that $v \in \Gamma$, and thus $v \notin V'$, a contradiction. This shows that G' is not connected, contradicting the f-connectivity of G. □

Now we can modify Procedure `disseminate` to require only $\lceil k/(fT) \rceil$ phases. Claim E.8 still holds, since it is only concerned with a single phase. The key change is in Claim E.9, which we now re-state as follows.

Claim E.13. *For each of the T smallest tokens $t \in I$ and phases i we have $|K_t^i(0)| \geq \min\{n, T \cdot f \cdot i\}$.*

Proof. Again by induction on i, with the base case being trivial. For the step, assume that $|K_t^i(0)| \geq T \cdot f \cdot i$. As argued in the proof of Claim E.9, at the end of phase $i+1$ we have $\Gamma^T(t) \subseteq K_t^{i+1}(0)$, where $\Gamma^T(t) := \{u \in V \mid \mathrm{tdist}_i(u, t) \leq T\}$. From Lemma E.12, $|\Gamma^T(t)| \geq \min\left\{n, |K_t^i(0)| + fT\right\}$, and the claim follows. □

Corollary E.14. *If $k \geq n$, then $K_t^{\lceil k/(fT) \rceil}(0) = V$ for each of the T smallest tokens $t \in I$.*

Proof. Because $fT \cdot \lceil k/(fT) \rceil \geq k$. □

By substituting the shortened `disseminate` in Algorithm 6, we obtain an algorithm that solves k-Committee in $O(n + n^2/(fT))$ time in $2T$-interval f-connected graphs.

E.4.2 Vertex Expansion

In this section, we show that if the communication graph is always an expander, the `disseminate` procedure requires $O(\lceil \log(n)/T \rceil)$ phases to disseminate the T smallest tokens.

Definition E.4. A static graph $G = (V, E)$ is said to have vertex expansion $\lambda > 0$ if for all $S \subseteq V$, if $|S| \leq \frac{|V|}{2}$ then $\frac{\Gamma(S)}{S} \geq 1 + \lambda$.

Definition E.5 (T-interval vertex expansion). A dynamic graph $G = (V, E)$ is said to have T-interval vertex expansion $\lambda > 0$ for $T \in \mathbb{N}$ if for all $r \in \mathbb{N}$, the static graph $G_{r,T} := \left(V, \bigcap_{i=r}^{r+T-1} E(r)\right)$ has vertex expansion λ.

Lemma E.15. *Let $G = (V, E)$, $|V| = n$ be a fixed undirected graph. If G has vertex expansion $\lambda > 0$, for any non-empty set $S \subseteq V$ and integer $d \geq 0$, we have*

$$|\Gamma^d(S)| \geq \begin{cases} \min\left\{(n+1)/2, |S| \cdot (1+\lambda)^d\right\} & \text{if } |S| \leq n/2 \\ n - |V \setminus S|/(1+\lambda)^d & \text{if } |S| > n/2. \end{cases}$$

Proof. The case $d = 0$ is trivial, the case $|S| \leq n/2$ follows directly from Definition E.4. For $|S| > n/2$, let $A = \Gamma^d(S) \setminus S$ and let $B = V \setminus (S \cup A)$. Note that any two nodes $u \in S$ and $v \in B$ are at distance at least $d + 1$. It therefore holds that $\Gamma^d(B) \subseteq V \setminus S$. Consequently, we have $\Gamma^d(B) < n/2$ and certainly also $|B| < n/2$ and thus by Definition E.4, $\Gamma^d(B) \geq |B|(1 + \lambda)^d$. Together, this implies that $n - |\Gamma^d(S)| = |B| \leq |V \setminus S|/(1 + \lambda)^d$ as claimed. □

Analogously to T-interval f-connected graphs, we can modify Procedure `disseminate` to require only $O(1 + \log_{1+\lambda}(n)/T)$ phases. Again, Claim E.8 still holds and the key is to restate Claim E.9, which now has to be adapted as follows.

Claim E.16. *We define $i_0 := \lceil \log_{1+\lambda}((n + 1)/2)/T \rceil$. For each of the T smallest tokens $t \in I$ and phases i, we have*

$$|K_t^i(0)| \geq \begin{cases} \min\left\{(n+1)/2, (1+\lambda)^{i \cdot T}\right\} & \text{for } i \leq i_0 \\ n - \frac{(n-1)/2}{(1+\lambda)^{(i-i_0)\cdot T}} & \text{for } i > i_0. \end{cases}$$

Proof. As in the other two cases, the proof is by induction on i, with the base case being trivial. Again, for the step, as argued in the proof of Claim E.9, at the end of phase $i + 1$ we have $\Gamma^T(t) \subseteq K_t^{i+1}(0)$, where $\Gamma^T(t) := \{u \in V \mid \mathrm{tdist}_i(u, t) \leq T\}$. The claim now immediately follows from Lemma E.15. □

Corollary E.17. *If $i \geq 2i_0 = O(1 + \log_{1+\lambda}(n))$, $K_t^i(0) = V$ for each of the T smallest tokens $t \in I$.* □

Consequently, in dynamic graphs with T-interval vertex expansion λ, n-gossip can be solved in $O(n + n \log_{1+\lambda}(n)/T)$ rounds.

E.5 Asynchronous Start

So far we assumed that all nodes begin executing the protocol in the same round. It is interesting to consider the case where computation is initiated by some subset of nodes, while the rest are asleep. We assume that sleeping nodes wake up upon receiving a message; however, since messages are delivered at the *end* of each round, nodes that are woken up in round r send their first message in round $r + 1$. Thus, nodes have no way of determining whether or not their messages were received by sleeping nodes in the current round.

Claim E.18. *Counting is impossible in 1-interval connected graphs with asynchronous start.*

Proof. Suppose by way of contradiction that $\mathcal{A}$ is a protocol for counting which requires at most $t(n)$ rounds in 1-interval connected graphs of size n. Let $n' = \max\{t(n) + 1, n + 1\}$. We will show that the protocol cannot distinguish a line of length n from a line of length n'.

Given a sequence $A = a_1 \circ \ldots \circ a_m$, let $\mathrm{shift}(A, r)$ denote the cyclic left-shift of A in which the first r symbols ($r \geq 0$) are removed from the beginning of the sequence and appended to the end. Consider an execution in a dynamic line of length n', where the line in round r is composed of two adjacent sections $A \circ B_r$, where $A = 0 \circ \ldots \circ (n - 1)$ remains static throughout the execution, and $B(r) = \mathrm{shift}(n \circ \ldots \circ (n' - 1), r)$ is left-shifted by one in every round. The computation is initiated by node 0 and all other nodes are initially asleep. We claim that the execution of the protocol in the dynamic graph $G = A \circ B(r)$ is indistinguishable in the eyes of nodes $0, \ldots, n - 1$ from an execution of the protocol in the static line of length n (that is, the network comprising section A alone). This is proven by induction on the round number, using the fact that throughout rounds $0, \ldots, t(n) - 1$ none of the nodes in section A ever receives a message from a node in section B: although one node in section B is awakened in every round, this node is immediately removed and attached at the end of section B, where it cannot communicate with the nodes in section A. Thus, the protocol cannot distinguish the dynamic graph A from the dynamic graph $A \circ B(r)$, and it produces the wrong output in one of the two graphs. □

If 2-interval connectivity is assumed, it becomes possible to solve gossip under asynchronous start. We begin by defining a version of the k-committee and k-verification problems that explicitly address sleeping nodes.

***k*-Commitee with Wakeup.** In the modified k-committee problem we require, as before, that no committee have more than k nodes. Sleeping nodes are not counted as belonging to any committee. In addition, if $k \geq n$, we require all nodes to be awake and to be in the same committee.

***k*-Verification with Wakeup.** In the modified k-verification problem, all awake nodes must eventually output 1 iff $k \geq n$. Sleeping nodes do not have to output anything. (Nodes that are awakened during the execution are counted as awake and must output a correct value; however, there is no requirement for the algorithm to wake up all the nodes.)

E.5.1 *k*-Verification with Wakeup

We modify the k-verification protocol as follows. First, each node that is awake at the beginning of the computation maintains a round counter c which is initialized to 0 and incremented after every round. Each message sent by the protocol carries the round counter of the sender, as well as a tag indicating that it is a k-verification protocol message (so that sleeping nodes can tell which protocol they need to join).

As before, each node u has a variable x_u which is initially set to its committee ID. In every round node u broadcasts the message $\langle k\text{-ver}, c_u, x_u \rangle$. If u hears a different committee ID or the special value $\perp$, it sets $x_u \leftarrow \perp$; if it hears a round counter greater than its own, it adopts the greater value as its own round counter. When a node u is awakened by receiving a message carrying the k-ver tag, it sets $x_u \leftarrow \perp$ and adopts the round counter from the message (if there is more than one message, it uses the largest one).

All awake nodes execute the protocol until their round counter reaches $2k$. At that point they halt and output 1 iff $x \neq \perp$.

$x \leftarrow committee$
$c \leftarrow 0$
while $c < 2k$ **do**
  broadcast $\langle k\text{-verif}, c, x \rangle$
  receive $\langle k\text{-verif}, c_1, x_1, \rangle, \ldots, \langle k\text{-verif}, c_s, x_s, \rangle$ from neighbors
  if $x_i \neq x$ for some $1 \leq i \leq s$ **then**
    $x \leftarrow \perp$
  $c \leftarrow \max\{c, c_1, \ldots, c_s\} + 1$
if $x = \perp$ **then**
  output 0
else
  output 1
upon awakening by receipt of messages $\langle k\text{-verif}, c_1, x_1, \rangle, \ldots, \langle k\text{-verif}, c_s, x_s, \rangle$**:**
  $x \leftarrow \perp$
  $c \leftarrow \max\{c_1, \ldots, c_s\} + 1$
upon awakening spontaneously (by the adversary):
  $x \leftarrow \perp$
  $c \leftarrow 0$

Algorithm 8: k-verification protocol with wakeup

Claim E.19. *Algorithm 8 solves the k-verification with wakeup problem if all nodes start in a state that represents a solution to k-committee with wakeup, and the graph is 2-interval connected.*

Proof. The case where $k \geq n$ is immediate: as in the synchronous start case, all nodes are awake at the beginning of the protocol, and no node ever hears a committee ID different from its own.

Suppose that $k < n$. Nodes that are awakened during the protocol set their x variable to $\perp$, so they will output 0; we only need to concern ourselves with nodes that are awake at the beginning and have a committee ID. We show that the size of each committee shrinks by at least one node every two rounds, so that at the end of the $2k$ rounds, all nodes have $x = \perp$.

Consider a cut between the nodes that belong to some committee C and still have $x = C$, and the rest of the nodes, which are either sleeping or have $x \neq C$. From 2-interval connectivity, some edge $\{u, v\}$ in the cut exists for the next two rounds. Assume that $x_u = C$. If v is asleep in the first round, wakes up when it receives u's message, and broadcasts $\perp$ in the second round. If v is awake in the first round it broadcasts $x_u \neq x_v$ in the first round. In both cases node u will change x_u to $\perp$ by the end of the second round. □

It remains to show that we can solve k-committee with asynchronous start. We can do this using the same approach as before, with one minor modification: as with k-verification, we maintain a round counter c at every node, and now each node u uses the pair $\langle c_u, u\rangle$ as its UID, instead of u alone. The pairs are ordered lexicographically, with *larger* round counters winning out over smaller ones; that is, $\langle c_u, u\rangle < \langle c_v, v\rangle$ iff $c_u > c_v$, or $c_u = c_v$ and $u < v$.

When a node receives a larger round counter than its own in a message, it adopts that value as its own round counter, and jumps to the appropriate part of the protocol (e.g., if the round counter it receives is $k + 3$, in the next round it will execute the fifth round of the invitation phase, because it knows that the first $k - 1$ rounds were taken up by the polling phase and the first four rounds of the invitation phase have passed already). We use round counters so that nodes that awaken during the execution of the protocol will know what the current round is, and to have the eventual leader be one of the nodes that woke up first.

Claim E.20. *Algorithm 6, when run with round counters and using pairs of the form $\langle c_u, u\rangle$ instead of UIDs, solves the k-committee with wakeup problem.*

Proof. First consider the case where $k \geq n$, and let u be the node with the smallest UID among the nodes that initiate the computation. The first polling phase executed by u lasts $k \geq n$ rounds, during which all nodes receive u's polling message and forward it, setting their round counter to match u's if it does not already. At the end of u's polling phase, all nodes are awake, all have the same round counter as u, and all have u as their leader. From this point on the execution proceeds as in the case of synchronous wakeup.

Next suppose that $k < n$. In this case we only need to show that no committee contains more than k members. But this, as always, is guaranteed by the fact that each committee contains only nodes invited by the node whose UID is the committee ID, and no node ever invites more than k nodes to join its committee. □

When nodes execute the full counting algorithm with asynchronous wakeup, different parts of the graph may be testing different values for k at the same time. However, the round counter serves to bring any lagging nodes up-to-date. When some node u first reaches $k \geq n$, even if other nodes are still testing smaller values for k, the first polling phase of u's k-committee instance will reach all nodes and cause them to join u's computation. (In fact they will join u's computation sooner, because to reach $k \geq n$ it had already had to go through at least $n-1$ rounds testing smaller values, so all nodes will have seen its current round already.)

E.6 Randomized Approximate Counting

We next show that under certain restrictions on the adversary providing the sequence of graphs, by using randomization, it is possible to obtain an approximation to the number of nodes in time almost linear in n with high probability, even if the dynamic graph is only 1-interval connected. The techniques we use are based on a gossiping protocol described in [32]. We assume that the nodes know some potentially loose upper bound N on n. When arguing about randomized algorithms, we need to specify which random choices the dynamic graph $G = (V, E)$ can depend on. We assume an adversary that is oblivious to all random choices of the algorithm.

Definition E.6 (Oblivious Adversary)**.** Consider an execution of a randomized algorithm $\mathcal{A}$. The dynamic graph $G = (V, E)$ provided by an oblivious adversary has to be independent of all random choices of $\mathcal{A}$.

In the sequel, we show that in the case of an oblivious adversary, it is possible to use randomization to efficiently compute an arbitrarily good estimate of n. In particular, we show that for any $\varepsilon > 0$, it is possible to compute an $(1+\varepsilon)$-approximation of n with high probability (in N) in time

- $O(n)$ when using messages of size $O(\log N \cdot (\log\log N + \log(1/\varepsilon))/\varepsilon^2)$
- $O(n \cdot (\log\log N + \log(1/\varepsilon))/\varepsilon^2)$ if the maximal message size is restricted to $O(\log N)$ bits.

For simplicity, we only describe the algorithm with slightly larger message sizes in detail and merely sketch how to adapt the algorithm if messages are restricted to $O(\log N)$ bits. For parameters $\varepsilon \in (0, 1/2)$ and $c > 0$, we define

$$\ell := \lceil (2+2c) \cdot 27 \ln(N)/\varepsilon^2 \rceil. \tag{1}$$

Initially, each node $v \in V$, computes ℓ independent exponential random variables $Y_1^{(v)}, \ldots, Y_\ell^{(v)}$ with rate 1. Following the aggregation scheme described in [32], we define

$$\forall S \subseteq V : \hat{n}(S) := \frac{\ell}{\sum_{i=1}^{\ell} \min_{v \in S} Y_i^{(v)}}. \tag{2}$$

If we choose a set S independently of the exponential random variables of the nodes, $\hat{n}(S)$ is a good estimate for the size of S as shown by the following lemma, which is proven in [32].

Lemma E.21 ([32])**.** *For every $S \subseteq V$ that is chosen independently of the random variables $Y_i^{(v)}$ for $i \in [\ell]$ and $v \in V$, we have*

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

Before describing the algorithm in detail, we give a brief overview. In order to obtain a good estimate for the total number of nodes n, the objective of each node will be to compute $\hat{n}(V)$ and thus $\min_{v \in V} Y_i^{(v)}$ for each $i \in [\ell]$. In each round, every node broadcasts the minimal Y_i value it has heard for every $i \in [\ell]$. If we assume that the sequence of graphs is chosen by an oblivious adversary, for each node $v \in V$ and round $r > 0$, $C_v(r)$ is independent of all the exponential random variables $Y_i^{(u)}$ chosen by nodes $u \in V$. Hence, as a consequence of Lemma E.21, $\hat{n}(C_v(r))$ is a good estimate of $|C_v(r)|$ for all r and v. Because $|C_v(r)| \geq r$ for all r and v (Claim E.2), each node can stop forwarding minimal Y_i values as soon as the value of $\hat{n}(C_v(r))$ exceeds the round number by a sufficient amount.

Executing the algorithm as described above would require the nodes to send exact values of exponential random variables, i.e., real values that cannot a priori be sent using a bounded number of bits. Therefore, each node $v \in V$ computes a rounded value $\tilde{Y}_i^{(v)}$ of $Y_i^{(v)}$ for each $i \in [\ell]$ as follows.

$$\tilde{Y}_i^{(v)} := \min\left\{ \frac{1}{4\ell N^{1+c}}, \max\left\{ \ln(4\ell N^{1+c}), \left(1+\frac{\varepsilon}{4}\right)^{\lfloor \log_{1+\varepsilon/4}(Y_i^{(v)}) \rfloor} \right\} \right\}. \tag{3}$$

Hence, $Y_i^{(v)}$ is rounded to the next smaller integer power of $1+\varepsilon/4$. Further, we restrict $\tilde{Y}_i^{(v)}$ to be within the range $[1/(4\ell N^{1+c}), \ln(4\ell N^{1+c})]$. We will show that with high probability, all variables $Y_i^{(v)}$ will be in this range and thus restricting the range only has an effect with negligible probability. As $\tilde{Y}_i^{(v)}$ is an integer

power of $1+\varepsilon/4$, it can be stored using $O(\log\log_{1+\varepsilon/4}(\ell N)) = O(\log\log N + \log(1/\varepsilon))$ bits. The details of the algorithm are given by Algorithm 9.

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

Algorithm 9: Randomized approximate counting in linear time, code for node v

Theorem E.22. *For $\varepsilon \in (0, 1/2)$ and $c > 0$, with probability at least $1 - 1/N^c$, every node of Algorithm 9 computes the same value $\tilde{n}_v(r) =: \tilde{n}$. Further $|\tilde{n} - n| \le \varepsilon n$.*

Proof. Let $\mathcal{A}$ be the event that the exponential random variables $Y_i^{(v)}$ for all $i \in [\ell]$ and $v \in V$ are within the range $[1/(4\ell N^{1+c}), \ln(4\ell N^{1+c})]$. For each $Y_i^{(v)}$, we have

$$\Pr\left(Y_i^{(v)} < \frac{1}{4\ell N^{1+c}}\right) = 1 - e^{-\frac{1}{4\ell N^{1+c}}} < \frac{1}{4\ell N^{1+c}}$$

and

$$\Pr\left(Y_i^{(v)} > \ln(4\ell N^{1+c})\right) = e^{\ln(4\ell N^{1+c})} = \frac{1}{4\ell N^{1+c}}.$$

As the number of random variables $Y_i^{(v)}$ is ℓn, we obtain $\Pr(\mathcal{A}) \ge 1 - 1/(2N^c)$ by a union bound.

Consider the state of some node $v \in V$ after $r > 0$ rounds. Because all minimal Z_i values are always forwarded, for all $i \in [\ell]$, it holds that $Z_i^{(v)} = \min_{u \in C_v(r)} \tilde{Y}_i^{(u)}$. In case of the event $\mathcal{A}$, for all i and v, we have

$$\tilde{Y}_i^{(v)} \le Y_i^{(v)} \le \left(1 + \frac{\varepsilon}{4}\right) \cdot \tilde{Y}_i^{(v)} \quad \text{and thus} \quad \tilde{n}_v(r) \ge \hat{n}(C_v(r)) \ge \frac{\tilde{n}_v(r)}{1+\varepsilon/4}. \tag{4}$$

We thus get

$$\begin{aligned}
&\Pr\left(\left(\left|\tilde{n}_v(r) - |C_v(r)|\right| > \varepsilon |C_v(r)|\right) \cap \mathcal{A}\right) \\
&\quad \overset{(\varepsilon \le 1/2)}{\le} \Pr\left(\left(\left|\tilde{n}_v(r) - |C_v(r)|\right| - \frac{\varepsilon}{4}|C_v(r)| > \left(1 + \frac{\varepsilon}{4}\right)\frac{2}{3}\varepsilon |C_v(r)|\right) \cap \mathcal{A}\right) \\
&\quad \overset{(4)}{\le} \Pr\left(\left(\left(1 + \frac{\varepsilon}{4}\right)\left|\hat{n}(C_v(r)) - |C_v(r)|\right| > \left(1 + \frac{\varepsilon}{4}\right)\frac{2}{3}\varepsilon |C_v(r)|\right) \cap \mathcal{A}\right) \\
&\quad \le \Pr\left(\left|\tilde{n}(C_v(r)) - |C_v(r)|\right| > \frac{2}{3}\varepsilon |C_v(r)|\right) \\
&\quad \overset{\text{(Lemma E.21)}}{\le} 2e^{-\varepsilon^2 \ell/27} \le 2e^{-2-(2+c)\ln N)} < \frac{1}{2N^{2+c}}.
\end{aligned}$$

In order to be able to apply Lemma E.21, we use that with an oblivious adversary, for all r and v, $C_v(r)$ is independent of all random variables $Y_i^{(u)}$. By applying a union bound, we obtain that with probability at

least $1 - 1/(2N^c)$ event $\overline{\mathcal{A}}$ occurs or

$$\forall v \in V, \forall r > 0 : \left|\tilde{n}_v(r) - |C_v(r)|\right| \leq \varepsilon \cdot |C_v(r)|. \tag{5}$$

Note that $C_v(r) = V$ for all $r \geq n - 1$ and that the union bound therefore is over $n(n-1) < N^2$ events. If (5) holds, we have

$$\tilde{n}_v(r) \geq (1 - \varepsilon) \cdot |C_v(r)| \geq (1 - \varepsilon) \cdot r$$

for all $r \leq n - 1$ and $v \in V$. Therefore, in this case no node terminates before round $n - 1$. Hence, all nodes get the same final value $\tilde{n}$ for $\tilde{n}_v(r)$ and by (5), it holds that $|\tilde{n} - n| \leq \varepsilon n$ as required. Because $\Pr(\overline{\mathcal{A}}) < 1/(2N^c)$, (5) holds with probability at least $1 - 1/N^c$ which completes the proof. □

F Lower Bounds for Token-Forwarding Algorithms

A token-forwarding algorithm for solving the gossip problem is an algorithm that does not manipulate the tokens in any way except storing and forwarding them. Specifically, the algorithm must satisfy the following conditions. Let $s_u^G(r)$ denote the message broadcast by node u in round r, when the algorithm is executed in dynamic graph $G = (V, E)$.

1. $s_u^G(r) \in \mathcal{T} \cup \{\perp\}$ for all round r and nodes u.
2. Nodes can only learn new tokens by receiving them, either in their input or in a message from another node. Formally, let $R_u^G(r) := \left\{s_v^G(r) \mid \{u, v\} \in E(r)\right\}$ denote the set of messages u receives in round r, and let

$$A_u^G(r) := I(u) \cup \left(\bigcup_{r'=0}^{r-1} R_u^G(r')\right).$$

   We require the following.

   - $s_u^G(r) \in A_u^G(r) \cup \{\perp\}$ for all nodes u and rounds r, and
   - If node u terminates in round r, then $A_u^G(r) = I$.

We omit the superscript G when it is obvious from the context.

F.1 $\Omega(n \log k)$ Lower Bound for Centralized k-Gossip in 1-Interval Connected Graphs

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. The lower bound holds as long as there is constant fraction of the nodes that still need to learn k^δ tokens for some positive constant δ.

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. 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 (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 F.1. *Any centralized algorithm for k-gossip in 1-interval connected graphs requires $\Omega(n \log k)$ rounds to complete in the worst case.*

Proof. We construct the communication graph for each round r in three stages.

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 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 u and v in different components must send different values, otherwise we would clearly have $t_u(r) \in A_v(r)$ and $t_v(r) \in A_u(r)$ and u and v 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 $\ell \geq 6$, 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, $Bottom$ according to the number of tokens they know, with nodes that know many tokens "on top": $Top := \{v_i \mid missing(v_i) \leq \ell/6\}$ and consequently $Bottom := \{v_i \mid missing(v_i) > \ell/6\}$.

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, and each bottom node will learn at most three new tokens. We begin by bounding the size of Top.

To that end, notice that $\sum_{u \in Top} missing(u) \geq \binom{|Top|}{2}$: 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. On the other hand, since each node in Top is missing at most $\ell/6$ tokens, it follows that $\sum_{u \in Top} missing(u) \leq |Top| \cdot (\ell/6)$. Putting the two facts together we obtain $|Top| \leq \ell/3 + 1$, and consequently also $|Bottom| = \ell - |Top| \geq 2\ell/3 - 1$.

Stage III: Connecting the nodes. The bottom nodes are relatively cheap to connect to, so we connect them in an arbitrary line. 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 using only the edges $P := \{\{u, v\} \mid u \in Top, v \in Bottom \text{ and } t_v \in A_u(r)\}$.

Since each top node is missing at most $\ell/6$ tokens, and each bottom node broadcasts a different value, for each top node there are at least $|Bottom| - \ell/6$ edges in P to choose from. But since we assume $\ell \geq 6$, $|Top| \leq \ell/3 + 1 \leq |Bottom| - \ell/6$; thus, each top node can be connected to a different bottom node using P-edges.

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

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

F.2 $\Omega(n + n^2/T)$ lower bound against knowledge-based token-forwarding algorithms

In this section we describe a lower bound against a restricted class of randomized token-forwarding algorithms. We represent randomness as a random binary string provided to each node at the beginning of the

execution. In every round, the nodes may consume a finite number of random bits, and use them to determine their message for that round and their next state. In every execution nodes only use finitely many coin tosses; we use an infinite string when modelling the algorithm in order to avoid

A token-forwarding algorithm is said to be *knowledge-based* if it can be represented as a collection of functions $\{f_u \mid u \in \mathcal{U}\} \subseteq \mathcal{P}(\mathcal{T})^* \times \{0,1\}^* \to D(\mathcal{T})$, such that in every round r, if R is the sequence of coin-tosses for node u up to round r (inclusive), the distribution according to which node u decides which token to broadcast is given by $f_u(A_u(0)\ldots, A_u(r), R)$.

We say that two dynamic graphs $G = (V, E)$ and $G' = (V', E')$ are *equal up to round* r if $V = V'$ and for all $r' < r$ we have $E(r') = E'(r')$. Let $D_u(r)$ denote the probability distribution for node u in round r. Knowledge-based algorithms have the following property.

Lemma F.2. *Let G, G' be two dynamic graphs that are equal up to round r, and let (V, I) be an instance of gossip. If u is a node such that $A_u^G(r) = I$, then for any round $r' \geq 0$ and string $R \in \{0,1\}^\omega$ we have $D_u^G(r', R) = D_u^{G'}(r', R)$.*

Proof. Since G and G' are equal up to round r, the sequences $A_u^G(0)\ldots A_u^G(r)$ and $A_u^{G'}(0)\ldots A_u^{G'}(r)$ are equal, and in particular $A_u^G(r) = A_u^{G'}(r) = I$.

By definition, for all $r' \geq r$ we have $A_u^G(r) \subseteq A_u^G(r')$ and $A_u^{G'}(r) \subseteq A_u^{G'}(r')$; therefore, $A_u^G(r') = A_u^{G'}(r') = I$ for all $r' \geq r$. Consequently, for all $r' \geq 0$, the sequences $A_u^G(0)\ldots A_u^G(r')$ and $A_u^{G'}(0)\ldots A_u^{G'}(r')$ are equal, and the claim follows. □

Theorem F.3. *Any knowledge-based token-forwarding algorithm for k-input gossip in T-interval connected graphs over n nodes requires $\Omega(n + nk/T)$ rounds to succeed with probability at least $1/2$. Further, if $|\mathcal{U}| = \Omega(n^2 k/T)$, then for sufficiently large n, deterministic algorithms require $\Omega(n + nk/T)$ rounds even when each node begins with at most one token.*

Proof. A lower bound of $\Omega(n)$ is demonstrated trivially in a static line network where at least one token starts at one end of the line. In the sequel we assume that $k > 1$.

Let $\{f_u\}$ be an knowledge-based token-forwarding algorithm for k-gossip. We use the UID space as the token domain, and choose nodes $u_1, \ldots, u_n$: for randomized algorithms we choose the UIDs arbitrarily, but for deterministic algorithms we must choose them carefully (see the last part of the proof). If the algorithm is randomized, we choose an input assignment where some node u_1 starts with all k tokens, and all other nodes $u_i \neq u_1$ start with a set $I(u_i) \subseteq \{u_1, u_i\}$. For deterministic algorithms, we later show that we can reach this state from some input assignment where each node starts with at most one token. For now let us suppose that we have reached some round r_0 in which $A_{u_1}(r_0) = I$ and for all $u_i \neq u_1$ we have $A_{u_i} \subseteq \{u_1, u_i\}$. In this starting state there are $n - 2$ nodes that do not know each token $t \neq u_1$. We abuse notation by using I to denote the set of all tokens $u_1, \ldots, u_k$ as well as the input assignment $I(u_i)$ to each node u_i.

Let $r_1 := r_0 + (n-2)(k-2)/(4T)$. For a token $t \in I$, let $\mathrm{E}[\#t]$ denote the expected number of times token t is broadcast by u between rounds r_0 and r_1 (exclusive). We have

$$\sum_{t \in I} \mathrm{E}[\#t] = \sum_{t \in I} \sum_{r=r_0+1}^{r_1-1} \Pr[t \text{ is broadcast in round } r] = r_1 - r_0 - 2 < (n-2)(k-2)/(4T).$$

Thus, there are at least two tokens $t \neq t'$ such that $\mathrm{E}[\#t], \mathrm{E}[\#t'] < (n-2)/(4T)$. Assume w.l.o.g. that $t \neq u_1$. From Markov's inequality, node u_1 broadcasts t less than $(n-2)/(2T)$ times with probability at least $1/2$ in any execution fragment starting from round r_0 and ending before round r_1, regardless of the dynamic graph we choose. The idea in the proof is to use u_1 as a buffer between the nodes that have already learned t and those that have not; since u_1 broadcasts t infrequently with high probability, in this manner we can limit the number of nodes that learn t.

We divide the rounds between r_0 and r_1 into segments $\alpha_1, \ldots, \alpha_m$. The graph remains static during each segment, but changes between segments. For each segment α_i we define two sets of nodes, C_i and D_i, where $C_i \cap D_i = \{u_1\}$. The nodes in D_i are "contaminated nodes" that might know token t at the beginning of the segment; we connect them in a clique. The nodes in C_i are "clean": initially, except for u_1, these nodes do not know t (some of them might learn t during the segment). The only way the nodes in C_i can learn t is if u_1 broadcasts it. In the first segment C_i is arranged in a line with u_1 at one end; in subsequent segments we "close" C_i to form a ring. Initially $D_1 = \{u_1, t\}$ and $C_1 = V \setminus \{t\}$ (recall that t, in addition to being a token, is also the UID of a node).

There are two types of segments in our construction.

- *Quiet* segments are ones in which u_1 does not broadcast t until the last round in the segment. In the last round of a quiet segment, u_1 broadcasts t, and some nodes in the ring become contaminated. The first segment α_1 is a quiet segment.
- After every quiet segment there follows one or more *active* segments, in which we clean up the ring and move contaminated nodes from C_i to D_i. We have to do this in a way that preserves T-interval connectivity. Each active segment is triggered by u_1 broadcasting t in the previous segment; if in some active segment u_1 does not broadcast t, the next segment will be quiet.

An active segment lasts exactly T rounds, and a quiet segment lasts until the first time u_1 broadcasts t (including that round).

Next we define in detail the construction of the communication graph in each segment. We maintain the following property:

($\star$) At the beginning of each active segment α_i, of all the nodes in C_i, only u_1 and at most T nodes in the T-neighborhood of u_1 in the ring know token t. Further, all the nodes that know t are on the same side of u_1. We refer to the side of u_1 where these nodes are located as the *contaminated side of* u_1.

($\star\star$) At the beginning of each quiet segment α_i, node u_1 is the only node in the ring that knows token t.

Let $v_1, \ldots, v_{n-2}$ be some ordering of the nodes in $C_1 \setminus \{u_1\}$ (nodes that initially do not know t). In each segment i the nodes in C_i will be some contiguous subset $v_{L_i}, \ldots, v_{R_i}$, where $L_{i+1} \geq L_i \geq 1$ and $R_{i+1} \leq R_i \leq n-2$ for all i. We place u_1 between v_{L_i} and v_{R_i} in the ring. Formally, the edges in any round $r \in \alpha_i$ where $i > 1$ are given by

$$E(r) := D_i^{(2)} \cup \{\{v_j, v_{j+1}\} \mid L_i \leq j < R_i\} \cup \{\{u_1, v_{L_i}\}, \{u_1, v_{R_i}\}\}.$$

In the first segment, the edges are $E(r) := D_1^{(2)} \cup \{\{v_j, v_{j_1}\} \mid 1 \leq j < n-2\} \cup \{\{u_1, v_1\}\}$ (we do not close the ring; this is to ensure that ($\star$) holds for the first active segment).

If α_i is a quiet segment, then we define $C_{i+1} := C_i$ (and consequently $D_{i+1} := D_i$); that is, the network does not change between α_i and α_{i+1} (except possibly for the closing of the ring after the first segment). However, if α_i is an active session, then u_1 has some neighbors in the ring that knows t, and they might spread t to other nodes even when u_1 does not broadcast t. We divide the nodes in $C_i \setminus \{u_1\}$ into three subsets.

- The *red nodes* red_i comprise the $2T$ nodes adjacent to u_1 on the contaminated side. The first T of these (the ones closer to u_1) may know t at the beginning of the segment; the other T may become contaminated if some of the first T broadcast token t. To be safe, we treat all red nodes as though they know t by the end of the session.
- The *yellow nodes* yellow_i comprise the T nodes adjacent to u_1 on the uncontaminated side. These nodes may learn t during the segment, but only if u_1 broadcasts it.
- The *green nodes* green_i are all the other nodes in the ring. These nodes cannot become contaminated during the segment, because their distance from any node that knows t is greater than T.

Our cleanup between segments α_i and α_{i+1} consists of moving all the red nodes into D_{i+1}. Formally, if $v_{L_i} \in \mathsf{red}_i$, then we define $v_{L_{i+1}} := v_{L_i} + 2T$ and $v_{R_{i+1}} := v_{R_i}$; otherwise, if $v_{R_i} \in \mathsf{red}_i$, then we define $v_{R_{i+1}} := v_{R_i}+2T$ and $v_{L_{i+1}} := v_{L_i}$. This satisfies ($\star$) and ($\star\star$): if u_1 does not broadcast t during segment α_i, then only the red nodes can know t at the end, and since we removed them from the ring, at the beginning of α_{i+1} no node knows t except u_1. The next segment will be quiet. Otherwise, if u_1 does broadcast t during α_i, then at the beginning of the next session (which is active) only the yellow nodes yellow_i can know t. These nodes then become red nodes in segment α_{i+1}, and there are T of them, as required.

The cleanup step preserves T-interval connectivity: assume that $\mathsf{red}_i = \{v_{L_i}, \ldots, v_{L_i+2T}\}$ (the other case is similar). Then the line $v_{L_i+2T}, v_{L_i+2T-1}, \ldots, u_1, v_{R_i}, v_{R_i+1}, \ldots, v_{L_i+2T-1}$ exists throughout both segment α_i and segment α_{i+1}: in segment α_i it exists as part of the ring, and in segment α_{i+1}, after we moved the red nodes into the clique D_{i+1}, the first part of the line $v_{L_i+2T}, v_{L_i+2T-1}, \ldots, u_1$ exists in the clique and the second part $u_1, v_{R_i}, v_{R_i+1}, \ldots, v_{L_i+2T-1}$ exists in the ring. The nodes in D_i are all connected to each other in both segments; thus, there is a static connected graph that persists throughout both segments α_i, α_{i+1}, and in particular it exists in any T rounds that start in α_i. (Note that α_{i+1} may be quiet, and in this case it can be shorter than T rounds. But in this case it will be followed by an active segment which has exactly the same edges and lasts T rounds.)

Notice that the number of uncontaminated nodes at the beginning of every active segment is at most $2T$ less than in the previous active session. Therefore the total number of nodes that know t by round r_1 is at most $2T$ times the number of active sessions, and this in turn is bounded by $2T$ times the number of rounds in which u_1 broadcasts t. Since u_1 broadcasts t less than $(n-2)/(2T)$ times with probability at least $1/2$, the algorithm is not finished by round r_1 with probability at least $1/2$.

Deterministic algorithms. If the algorithm is deterministic, we first show that there exists an input assignment in which each node begins with at most one token, from which either

1. the algorithm runs for $\Omega(nk/T)$ rounds, or
2. we reach a round r_0 in which some node u_1 has $A_{u_1}(r_0) = I$ and for all $i \neq 1$ we have $A_{u_i}(r_0) \subseteq \{u_1, u_i\}$.

In the case of (2), we then continue with the same proof as for the input assignment where some node starts with all tokens and the rest of the nodes have no tokens (see above). Since we are free to choose the input assignment, we restrict attention to instances in which the inputs to k nodes are their own UIDs, and the inputs to the other tokens are $\emptyset$.

For deterministic algorithms the function f_u representing node u's behavior must return a distribution in which one token has probability 1. We abuse notation slightly by using $f_u(A_u(0) \ldots, A_u(r-1))$ to denote this token.

We say that a process $u \in \mathcal{U}$ *fires in round* r if when process u receives $\{u\}$ as its input and hears nothing in the first $r-1$ rounds, it will stay silent in those rounds and then spontaneously broadcast its token in round r. Formally, process u fires in round r if

1. For all $r' < r$ we have $f_u(\{\perp\}^{r'}) = \perp$, and
2. $f_u(\{u\}^r) = u$.

If process u does not fire in any round $r' \leq r$, we say that u is *passive until round* r. (Note that nodes that receive no tokens in their input have no choice but to broadcast nothing until they receive a token from someone.)

Since $|\mathcal{U}| = \Omega(n^2k/T)$, there exist constants c, n_0 such that for all $n \geq n_0$ we have $|\mathcal{U}| \geq cn^2k + n - 1$. Let $n \geq n_0$. We divide into two cases.

Case I. There exist $u_1, \ldots, u_n \in \mathcal{U}$ that are all passive until round cnk/T. In this case we construct the static clique over $u_1, \ldots, u_n$ and let the algorithm run. During the first cnk/T rounds, all nodes send only

$\perp$, and no node learns new tokens. Consequently all nodes u_i have $A_{u_i}(nk/T) = \mathit{in}(u_i) \neq I$, and the algorithm cannot terminate by round cnk/T.

Case II. All but $n-1$ processes fire no later than round cnk/T.

Since $|\mathcal{U}| \geq c(n^2k/T + n - 1)$, by the pigeonhole principle there must exist a round $r_0 \leq cnk/T$ such that at least n processes fire in round r_0. Let $u_1, \ldots, u_n$ be n such processes. We choose the instance where each node u_i receives as input $\{u_i\}$ if $i \leq k$, or $\emptyset$ if $i > k$.

Let S be the static star with u_1 at the center: $S = (V, E_S)$, where $E_S(r) = \{\{u_1, u_i\} \mid i > 1\}$ for all r. Because all nodes fire in round r_0, when the algorithm is executed in S, the network is silent until round r_0. In round r_0 all nodes that have a token broadcast it. Following round r_0 we have $A_{u_1}(r_0 + 1) = I$, and for all $i > 1$, $A_{u_i}(r_0 + 1) = I(u_i) \cup \{u_1\} \subseteq \{u_1, u_i\}$. This is the state from which we start the main body of the proof above. $\square$