Computer Science and Artificial Intelligence Laboratory
Technical Report

MIT-CSAIL-TR-2009-058 November 10, 2009

Distributed Computation in Dynamic Networks

Fabian Kuhn, Nancy Lynch, and Rotem Oshman

massachusetts institute of technology, cambridge, ma 02139 usa — www.csail.mit.edu

Distributed Computation in Dynamic Networks : Technical Report

Fabian Kuhn — fabian.kuhn@usi.ch
Nancy Lynch — lynch@csail.mit.edu
Rotem Oshman — rotem@csail.mit.edu

Faculty of Informatics, University of Lugano, 6904 Lugano, Switzerland
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 -*interval connectivity* (for), which stipulates that for every consecutive rounds there exists a stable connected spanning subgraph. For 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 rounds using messages of size , where is the size of the input to a single node. Further, if the graph is -interval connected for , the computation can be sped up by a factor of , and any function can be computed in rounds using messages of size . We also give two lower bounds on the gossip problem, which requires the nodes to disseminate pieces of information to all the nodes in the network. We show an bound on gossip in 1-interval connected graphs against centralized algorithms, and an bound on exchanging pieces of information in -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 *-interval connectivity*: throughout every block of 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 bits per message.

To demonstrate the power of the adversary in the dynamic graph model, consider the problem of *local token circulation*: each node has a local Boolean variable , and if , node 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 has the token, and provide that node with only one edge . Node then has no choice except to eventually pass the token to . After receives it, the adversary can turn around and remove all of 's edges except , so that has no choice except to pass the token back to . In this way the adversary can prevent the token from ever visiting any node except .

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.

-*gossip*, in which pieces of information, called *tokens*, are handed out to some nodes in the network, and all nodes must collect all tokens.

We are especially interested in the variant of -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 rounds in -interval connected graphs. Then we extend the algorithm for -interval connected graphs with known , obtaining an -round protocol for counting or all-to-all gossip. When is not known, we show that both problems can be solved in 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 lower bound on -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 lower bound on -gossip in -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 -gossip and counting even when computation is initiated by one node and the rest of the nodes are asleep.

Related work. For static networks, information dissemination and basic network aggregation tasks have been extensively studied (see e.g. [5, 16, 29]). In particular, the -gossip problem is analyzed in [35], where it is shown that tokens can always be broadcast in time 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, 23]) or multiple messages [11, 26] 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 [17, 19, 21]; gossiping aggregation protocols that can be used to approximate the size of the system are described in [20, 31]. 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, 29]. 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, 36, 18]), 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 [22, 24, 30], 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 linkreversal [14], 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 [15, 27, 28]. Most closely related to the problems studied here is [32], where a few basic results in a similar setting are proved; mainly it is shown that in -interval connected dynamic graphs (the definition in [32] 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 -interval connectivity was used in [25], where two of the authors studied clock synchronization in *asynchronous* dynamic networks. In [25] it is assumed that the network satisfies -interval connectivity for a small value of , 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. [33, 34].

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 Network Model

2.1 Dynamic Graphs

A synchronous dynamic network is modelled by a dynamic graph , where is a static set of nodes, and is a function mapping a round number to a set of undirected edges . Here is the set of all possible undirected edges over .

Definition 2.1 (-Interval Connectivity)**.** A dynamic graph is said to be -*interval connected* for if for all , the static graph is connected. If is -interval connected we say that is *always connected.*

Definition 2.2 (-Interval Connectivity)**.** A dynamic graph is said to be -*interval connected* if there exists a connected static graph such that for all , .

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 -interval connectivity assumption is then replaced by -*interval strong connectivity*, which requires that be strongly connected (where 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 2.3 (Causal Order)**.** Given a dynamic graph , we define an order , where iff and . The *causal order* is the reflexive and transitive closure of . We also write if there exists some such that .

Definition 2.4 (Influence Sets)**.** We denote by the set of nodes whose state in round causally influences node in round . We also use the short-hand .

2.2 Communication and Adversary Model

Nodes communicate with each other using *anonymous broadcast*, with message sizes limited to . At the beginning of round , each node 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 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 all messages broadcast by nodes such that . 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.

2.3 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 sends its first message in round .

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

2.4 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 on the size of the network. The upper bound is assumed to be bounded by some function of the true size, e.g., .

Exact size: nodes know the size of the network.

2.5 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 to denote the value of node 's local variable at the beginning of round , and to denote the input to node .

3 Problem Definitions

We assume that nodes have unique identifiers (UIDs) from some namespace . Let be a problem domain. Further, let denote the set of all partial functions from to .

A *problem* over is a relation , such that if then is finite and . Each instance induces a set of nodes, and we say that an algorithm *solves* instance if in any dynamic graph , when each node starts with as its input, eventually each node outputs a value such that .

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

is finite and

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

- and iff

-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 outputs a value , and we require the following properties.

1. (“Safety”) The size of each committee is at most , that is, for all we have .
2. (“Liveness”) If then all nodes in the graph join one committee, that is, for all we have .

-Gossip. The gossip problem is defined over a token domain . Each node receives in its input a set of tokens, and the goal is for all nodes to output all tokens. Formally,

- is finite and

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

All-to-All gossip: instances where for all we have .

-gossip with known : in this variant nodes know , i.e., they receive as part of the input.

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

4 Relationships

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

4.1 -Committee -Verification

Claim 4.1. *-verification reduces to -committee.*

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

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

Claim 4.2. *-committee reduces to -verification.*

Proof. Again, suppose the nodes are initially in a state that represents a solution to -verification: they have a Boolean flag which is set to 1 iff . We solve -committee as follows: if , then each node outputs its own ID as its committee ID. This is a valid solution because when the only requirement is that no committee have more than nodes. If , then for 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 indicates that , after 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. □

4.2 Counting vs. -Verification

Since we can solve -verification in time in -interval connected graphs, we can find an upper bound on the size of the network by checking whether for values of starting from 1 and doubling with every wrong guess. We know how to verify whether in time, and hence the time complexity of the entire procedure is . Once we establish that for some value of , to get an actual count we can then go back and do a binary search over the range (recall that , otherwise we would not have reached the current value of).

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

4.3 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. -gossip (with unknown).

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

5 Upper Bounds

In this section we give algorithms for some of the problems introduced in Section 3, 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 -committee,
3. Solving -committee allows us to solve -verification and therefore also counting (see Section 4).

We initially focus on the case of synchronous start. The modifications necessary to deal with asynchronous start are described in Section 5.5.

5.1 Always-Connected Graphs

5.1.1 Basic Information Dissemination

It is a basic fact that in 1-interval connected graphs, a single piece of information requires at most rounds to reach all the nodes in the network, provided that it is forwarded by all nodes that receive it. Formally, let denote the set of nodes that has "reached" by round . If knows a token and broadcasts it constantly, and all other nodes broadcast the token if they know it, then all the nodes in know the token by round .

Claim 5.1. *For any node and round we have .*

Proof. By induction on . For the claim is immediate. For the step, suppose that , and consider round . If then the claim is trivial, because . Thus, suppose that . Since is connected, there is some edge in the cut . From the definition of the causal order we have , and therefore . ☐

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 rounds. Note, however, that in this case nodes do not necessarily *know* when they know the smallest or biggest token.

5.1.2 Counting in linear time with -bit messages

We begin by describing a linear-time counting/ -gossip protocol which uses messages of size . 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 (-interval connectivity) which allows nodes to communicate with at least one of their neighbors for at least rounds.

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

```
for              do
|   broadcast
|   receive              from neighbors
|
|   if         then  terminate and output
end
```

Algorithm 1: Counting in linear time using large messages

Claim 5.2. *For any node and rounds we have .*

Proof. By induction on . For the claim is immediate.

Suppose that for all nodes and rounds such that and we have . Let be two rounds such that .

If then we are done, because . Thus, assume that . Since the communication graph in round is connected, there is some edge such that and . We have , and consequently and . Also, from the induction hypothesis, . Together we obtain , as desired. ☐

Claim 5.3. *For any node and round we have .*

Proof. It is easily shown that for all we have . From the previous claim we have for all , and the claim follows. ☐

The correctness of the protocol follows from Claim 5.3: suppose that for some round and node we have . From Claim 5.3, then, . Applying the claim again, we see that , and since for all , we obtain . This shows that nodes compute the correct count. For termination we observe that the size of never exceeds , so all nodes terminate no later than round .

5.1.3 -committee with -bit messages

We can solve -committee in rounds as follows. Each node stores a local variable in addition to . 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 cycles, each consisting of two phases, *polling* and *selection*.

1. Polling phase: for 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 , where is the ID of the leader that issued the invitation, and is the ID of the invited node. All nodes propagate the smallest invitation of which they are aware for (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 cycles, any node that has not been invited to join a committee outputs .

```
for            do
    // Polling phase
    if             then
        _          ;    // The node nominates itself for selection
    else
        _
    end
    for            do
        broadcast  _
        receive         from neighbors
            _               _
    end
    // Update leader
                         _
    // Selection phase
    if           then
        // Leaders invite the smallest ID they heard
                            _
    else
        // Non-leaders do not invite anybody

    end
    for            do
        broadcast
        receive         from neighbors
                                      ;      // (in lexicographic
        order)
    end
    // Join the leader's committee, if invited
    if                      then

    end
end
if              then

end
```

Algorithm 2: -committee in always-connected graphs

Claim 5.4. *The protocol solves the -committee problem.*

Proof. We show that after the protocol ends, the values of the local variables constitute a valid solution to -committee.

1. In each cycle, each node invites at most one node to join its committee. After cycles at most nodes have joined any committee. Note that the first node invited by a leader to join 's committee is always itself. Thus, if after cycles node has not been invited to join a committee, it follows that 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 , and let be the node with the smallest ID in the network. Following the polling phase of the first cycle, all nodes have for the remainder of the protocol. Thus, throughout the execution, only node issues invitations, and all nodes propagate 's invitations. Since rounds are sufficient for to hear the ID of the minimal active node in the network, in every cycle node successfully identifies this node and invites it to join 's committee. After cycles, all nodes will have joined.

□

Remark. The protocol can be modified easily to solve -gossip if . Let be the token node received in its input (or if node did not receive a token). Nodes attach their tokens to their IDs, and send pairs of the form instead of just . Likewise, invitations now contain the token of the invited node, and have the structure . 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.

5.2 -interval Connected Graphs

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

While executing this protocol, nodes keep track of the current round number (starting from zero). When a node reaches a round in which , it terminates and outputs as the count.

for **do**
if **then**
broadcast
end
receive from neighbors
if **then** terminate and output
end
return

Algorithm 3: Counting in -interval connected graphs

5.2.1 Analysis

Let denote the shortest-path distance between and in the stable subgraph , and let denote the -neighborhood of in , that is, . We use and to denote the values of local variables and at node in the beginning of round . Note the following properties:

1. for all and .
2. If and are neighbors in , then for all , because every value sent by is received by and added to .
3. and are monotonic, that is, for all and we have and .

Claim 5.5. *For every two nodes and round such that , either or .*

Proof. By induction on . For the claim is immediate.

Suppose the claim holds for round , and consider round . Let be nodes such that ; we must show that either or .

If , then the claim holds: is broadcast in the first round, and thereafter we have for all .

Otherwise, let be a neighbor of along the shortest path from to in ; that is, is a neighbor of such that . Since we have .

From the induction hypothesis on and in round , either or . Applying property 2 above, this implies the following.

( ) Either or .

If or then we are done, because . Suppose then that and . It is sufficient to prove that : this shows that in round node broadcasts and adds it to , yielding and proving the claim.

We show this using . If , then , because we assumed that . Otherwise states that , and since we assumed that , this again shows that . ☐

Claim 5.6. *If , then for all nodes we have .*

Proof. Let . For any node , Claim 5.5 shows that either or . Thus, either or . Since and is connected we have , and therefore in both cases we have . ☐

Claim 5.7. *The algorithm terminates in linear time and outputs the correct count at all nodes.*

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

Correctness follows from Claim 5.6. Suppose that in round node has , and let . We must show that .

From Claim 5.6, if then . By definition of we have and hence from Property 3 we obtain , which is not the case. Thus, and . Applying the same reasoning as in Claim 5.6 to round , we see that either or . Since the first cannot occur it must be the case that , and we are done. ☐

5.3 Finite-Interval Connected Graphs

Next we generalize the protocol above, in order to solve -committee in -interval connected graphs. The general protocol requires rounds (and assumes that 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 nodes and disseminates their IDs throughout the network. We generalize and refine Claim 5.5 for the case where there are initially up to tokens, but only the smallest tokens need to be disseminated.

5.3.1 -gossip in -interval connected graphs

The "pipelining effect" we used in the -interval connected case allows us to disseminate tokens in rounds, given that the graph is -interval connected. The idea is to use a similar protocol to the -interval connected case, except that the protocol is "restarted" every rounds: all nodes empty the set (but not), which causes them to re-send the tokens they already sent, starting from the smallest and working upwards. The 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 . The input at each node is an initial set of tokens. In addition, it is assumed that all nodes have a common guess for the size of the network. The protocol guarantees that the smallest tokens in the network are disseminated to all nodes, provided that the graph is -interval connected and that .

```
for                          do
    for                 do
        if         then
            broadcast
        end
        receive            from neighbors
    end
end
return
```

Function `disseminate( )`

We refer to each iteration of the inner loop as a *phase*. Since a phase lasts rounds and the graph is -interval connected, there is some connected subgraph that exists throughout the phase. Let be a connected subgraph that exists throughout phase , for . We use to denote the distance between nodes in .

Let denote the set of nodes that know token by the beginning of round , that is, . In addition, let be the set of smallest tokens in . Our goal is to show that when the protocol terminates we have for all .

For a node , a token , and a phase , we define to be the distance of from the nearest node in that knows at the beginning of phase :

Here and in the sequel, we use the convention that . For convenience, we use to denote the value of in round of phase . Similarly we denote and .

The following claim characterizes the spread of each token in each phase. It is a generalization of Claim 5.5, and the proof is similar.

Claim 5.8. *For any node , token and round such that , either or includes at least tokens that are smaller than .*

Proof. By induction on . For the claim is immediate.

Suppose the claim holds for round of phase , and consider round . If , then and the claim holds trivially. Thus, suppose that . Hence, , and the induction hypothesis applies: either or includes at least tokens that are smaller than . In the first case we are done, since ; thus, assume that , and includes at least tokens smaller than . However, if includes at least tokens smaller than , then so does , and the claim is again satisfied; thus we assume that includes *exactly* tokens smaller than .

It is sufficient to prove that : if this holds, then in round node broadcasts , which is either or a token smaller than ; thus, either or includes at least tokens smaller than , and the claim holds.

First we handle the case where . In this case, . Since we assumed that we have , which implies that .

Next suppose that . Let be a node such that (such a node must exist from the definition of), and let be a neighbor of along the path from to in , such that . From the induction hypothesis, either or includes at least tokens that are smaller than . Since the edge between and exists throughout phase , node receives everything sends in phase , and hence . Finally, because we assumed that contains exactly tokens smaller than , and does not include itself, we have , as desired. □

Claim 5.9. *For each of the smallest tokens and phases , we have .*

Proof. The proof is by induction on . For the claim is immediate. For the induction step, suppose that , and consider phase .

Let denote the -neighborhood of , that is, . From Claim 5.8 applied to round of phase , for all , either or includes at least tokens smaller than . Since is one of the smallest tokens in the network, this latter case is impossible. Thus, every node has , which implies that . In addition, , because nodes never forget tokens they have learned.

Since is connected, . Combining with the induction hypothesis we obtain , and the claim follows. □

Procedure `disseminate` terminates at the end of phase , or, equivalently, at the beginning of phase . By this time, if the guess for the size of the network was correct, all nodes have learned the smallest tokens.

Corollary 5.10. *If , then for each of the smallest tokens .*

Proof. The claim follows from Claim 5.9, because . □

5.3.2 -committee in -interval connected graphs

We can solve the -committee problem in rounds using Algorithm 5. The idea is similar to Algorithm 2, except that leaders invite 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 .

```
for                     do
|  if               then
|  |          ;         // The node nominates itself for selection
|  else
|  |
|  end
|         disseminate
|
|  if             then
|  |  // Leaders invite the   smallest IDs they collected
|  |  // (or less in the final cycle, so that the total does not
|  |     exceed  )
|  |  if             then
|  |  |     smallest- (      )
|  |  else
|  |  |
|  |  |     smallest- (      )
|  |  end
|  else
|  |  // Non-leaders do not invite anybody
|  |
|  end
|         disseminate
|  // Join the leader's committee, if invited
|  if                      then
|  |
|  end
end
if                then
|
end
```

Algorithm 5: -committee in -interval connected graphs

Claim 5.11. *The protocol above solves -committee in rounds.*

5.3.3 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 -interval connected, invitations may not reach their destination, and the committees formed may contain less than nodes even if . However, even when the graph is not -interval connected, no committee contains *more* than nodes, simply because no node ever issues more than invitations. Thus, if nodes guess a value for and use the -committee protocol above to solve -verification, their error is one-sided: if their guess for is too large they may falsely conclude that when in fact , but they will never conclude that when .

This one-sided error allows us to try different values for and without fear of mistakes. We can count in time in graphs where is *unknown* using the following scheme. I assume the version of -verification that returns the set of all nodes if , or the special value if .

for **do**
  for **do**
    if *-verification assuming* *-interval connectivity returns* **then**
      return
    end
  end
end

Algorithm 6: Counting in in -interval connected graphs where is unknown

The time required for -verification assuming -interval connectivity is for all , and thus the total time complexity of the -th iteration of the outer loop is .

If the communication graph is -interval connected, the algorithm terminates the first time we reach values of and such that and . Let be the smallest power of 2 that is no smaller than ; clearly . Let us show that the algorithm terminates when we reach .

First consider the case where , and hence . When we reach the last iteration of the inner loop, where , we try to solve -verification assuming -interval connectivity. This must succeed, and the algorithm terminates.

Next, suppose that . Consider the iteration of the inner loop in which . In this iteration, we try to solve -verification assuming -interval connectivity. Since , 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 rounds.

The asymptotic time complexity of this algorithm only improves upon the original algorithm (which assumes only 1-interval connectivity) when . 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. 6 is executed in odd rounds. This will lead to a time complexity of , because we terminate when either algorithm returns a count.

5.4 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 factor.

5.4.1 -Connected Graphs

Definition 5.1. A static graph is *-connected* for if the removal of any set of at most nodes from does not disconnect it.

Definition 5.2 (-interval -connectivity)**.** A dynamic graph is said to be *-interval -connected* for if for all , the static graph is -connected.

Definition 5.3 (Neighborhoods)**.** Given a static graph and a set of nodes, the *neighborhood* of in is the set . The -*neighborhood* of is defined inductively, with and for . We omit the subscript when it is obvious from the context.

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

Lemma 5.12 (Neighborhood Growth)**.** *If is a static -connected graph, then for any non-empty set and integer , we have .*

Proof. By induction on . For the claim is immediate. For the step, suppose that . Suppose further that , otherwise the claim is immediate. This also implies that , because . Thus the induction hypothesis states that .

Let denote the "new" nodes in the -neighborhood of . It is sufficient to show that , because then , and we are done.

Suppose by way of contradiction that , and let be the subgraph obtained from by removing the nodes in . Because is -connected and , the subgraph is connected. Consider the cut in . Because and , we have , and because and , we also have . However, the cut is empty: if there were some edge such that and , then by definition of we would have . This in turn would imply that , and thus , a contradiction. This shows that is not connected, contradicting the -connectivity of . □

Now we can modify Procedure `disseminate` to require only phases. Claim 5.8 still holds, since it is only concerned with a single phase. The key change is in Claim 5.9, which we now re-state as follows.

Claim 5.13. *For each of the smallest tokens and phases we have* .

Proof. Again by induction on , with the base case being trivial. For the step, assume that . As argued in the proof of Claim 5.9, at the end of phase we have , where . From Lemma 5.12, , and the claim follows. □

Corollary 5.14. *If , then for each of the smallest tokens .*

Proof. Because . □

By substituting the shortened `disseminate` in Algorithm 5, we obtain an algorithm that solves -Committee in time in -interval -connected graphs.

5.4.2 Vertex Expansion

In this section, we show that if the communication graph is always an expander, the `disseminate` procedure requires phases to disseminate the smallest tokens.

Definition 5.4. A static graph is said to have vertex expansion if for all , if — then — .

Definition 5.5 (-interval vertex expansion)**.** A dynamic graph is said to have -interval vertex expansion for if for all , the static graph has vertex expansion .

Lemma 5.15. *Let , be a fixed undirected graph. If has vertex expansion , for any non-empty set and integer , we have*

if
if

Proof. The case is trivial, the case follows directly from Definition 5.4. For , let and let . Note that any two nodes and are at distance at least . It therefore holds that . Consequently, we have and certainly also and thus by Definition 5.4, . Together, this implies that as claimed. □

Analogously to -interval -connected graphs, we can modify Procedure `disseminate` to require only phases. Again, Claim 5.8 still holds and the key is to restate Claim 5.9, which now has to be adapted as follows.

Claim 5.16. *We define . For each of the smallest tokens and phases , we have*

for
for

Proof. As in the other two cases, the proof is by induction on , with the base case being trivial. Again, for the step, as argued in the proof of Claim 5.9, at the end of phase we have , where . The claim now immediately follows from Lemma 5.15. ☐

Corollary 5.17. *If , for each of the smallest tokens .* ☐

Consequently, in dynamic graphs with -interval vertex expansion , -gossip can be solved in rounds.

5.5 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 send their first message in round . Thus, nodes have no way of determining whether or not their messages were received by sleeping nodes in the current round.

Claim 5.18. *Counting is impossible in 1-interval connected graphs with asynchronous start.*

Proof. Suppose by way of contradiction that is a protocol for counting which requires at most rounds in 1-interval connected graphs of size . Let . We will show that the protocol cannot distinguish a line of length from a line of length .

Given a sequence , let denote the cyclic left-shift of in which the first symbols ( ) are removed from the beginning of the sequence and appended to the end. Consider an execution in a dynamic line of length , where the line in round is composed of two adjacent sections , where remains static throughout the execution, and is left-shifted by one in every round. The computation is initiated by node and all other nodes are initially asleep. We claim that the execution of the protocol in the dynamic graph is indistinguishable in the eyes of nodes from an execution of the protocol in the static line of length (that is, the network comprising section alone). This is proven by induction on the round number, using the fact that throughout rounds none of the nodes in section ever receives a message from a node in section : although one node in section is awakened in every round, this node is immediately removed and attached at the end of section , where it cannot communicate with the nodes in section . Thus, the protocol cannot distinguish the dynamic graph from the dynamic graph , 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 -committee and -verification problems that explicitly address sleeping nodes.

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

-Verification with Wakeup. In the modified -verification problem, all awake nodes must eventually output 1 iff . 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.)

5.5.1 -Verification with Wakeup

We modify the -verification protocol as follows. First, each node that is awake at the beginning of the computation maintains a round counter 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 -verification protocol message (so that sleeping nodes can tell which protocol they need to join).

As before, each node has a variable which is initially set to its committee ID. In every round node broadcasts the message **-** . If hears a different committee ID or the special value , it sets ; if it hears a round counter greater than its own, it adopts the greater value as its own round counter. When a node is awakened by receiving a message carrying the **-** tag, it sets 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 . At that point they halt and output iff .

while **do**
broadcast -
receive - , . . . , - from neighbors
if for some **then**
|
end

end
if **then**
output 0
else
output 1
end
upon awakening by receipt of messages - **, . . . ,** - **:**

upon awakening spontaneously (by the adversary):

Algorithm 7: -verification protocol with wakeup

Claim 5.19. *Algorithm 7 solves the -verification with wakeup problem if all nodes start in a state that represents a solution to -committee with wakeup, and the graph is 2-interval connected.*

Proof. The case where 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 . Nodes that are awakened during the protocol set their variable to , 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 rounds, all nodes have .

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

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

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 , in the next round it will execute the fifth round of the invitation phase, because it knows that the first 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 5.20. *Algorithm 5, when run with round counters and using pairs of the form instead of UIDs, solves the -committee with wakeup problem.*

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

Next suppose that . In this case we only need to show that no committee contains more than 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 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 at the same time. However, the round counter serves to bring any lagging nodes up-to-date. When some node first reaches , even if other nodes are still testing smaller values for , the first polling phase of 's -committee instance will reach all nodes and cause them to join 's computation. (In fact they will join 's computation sooner, because to reach it had already had to go through at least rounds testing smaller values, so all nodes will have seen its current round already.)

5.6 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 with high probability, even if the dynamic graph is only -interval connected. The techniques we use are based on a gossiping protocol described in [31]. We assume that the nodes know some potentially loose upper bound on . When arguing about randomized algorithms, we need to specify which random choices the dynamic graph can depend on. We assume an adversary that is oblivious to all random choices of the algorithm.

Definition 5.6 (Oblivious Adversary)**.** Consider an execution of a randomized algorithm . The dynamic graph provided by an oblivious adversary has to be independent of all random choices of .

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 . In particular, we show that for any , it is possible to compute an -approximation of with high probability (in) in time

when using messages of size

if the maximal message size is restricted to 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 bits. For parameters and , we define

(1)

Initially, each node , computes independent exponential random variables with rate . Following the aggregation scheme described in [31], we define

(2)

If we choose a set independently of the exponential random variables of the nodes, is a good estimate for the size of as shown by the following lemma, which is proven in [31].

Lemma 5.21 ([31])**.** *For every that is chosen independently of the random variables for and , we have*

| | −

Before describing the algorithm in detail, we give a brief overview. In order to obtain a good estimate for the total number of nodes , the objective of each node will be to compute and thus for each . In each round, every node broadcasts the minimal value it has heard for every . If we assume that the sequence of graphs is chosen by an oblivious adversary, for each node and round , is independent of all the exponential random variables chosen by nodes . Hence, as a consequence of Lemma 5.21, is a good estimate of for all and . Because for all and (Claim 5.2), each node can stop forwarding minimal values as soon as the value of 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 computes a rounded value of for each as follows.

− (3)

Hence, is rounded to the next smaller integer power of . Further, we restrict to be within the range . We will show that with high probability, all variables will be in this range and thus restricting the range only has an effect with negligible probability. As is an integer power of , it can be stored using bits. The details of the algorithm are given by Algorithm 8.

for **do**
    broadcast
    receive from neighbors
    for **do**
    end
    if **then** terminate and output
end

Algorithm 8: Randomized approximate counting in linear time, code for node

Theorem 5.22. *For and , with probability at least , every node of Algorithm 8 computes the same value . Further .*

Proof. Let be the event that the exponential random variables for all and are within the range . For each , we have

and

As the number of random variables is , we obtain by a union bound.

Consider the state of some node after rounds. Because all minimal values are always forwarded, for all , it holds that . In case of the event , for all and , we have

and thus (4)

We thus get

(4)

Lemma 5.21

In order to be able to apply Lemma 5.21, we use that with an oblivious adversary, for all and , is independent of all random variables . By applying a union bound, we obtain that with probability at least event occurs or

(5)

Note that for all and that the union bound therefore is over events. If (5) holds, we have

for all and . Therefore, in this case no node terminates before round . Hence, all nodes get the same final value for and by (5), it holds that as required. Because , (5) holds with probability at least which completes the proof. □

6 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 denote the message broadcast by node in round , when the algorithm is executed in dynamic graph .

1. for all round and nodes .
2. Nodes can only learn new tokens by receiving them, either in their input or in a message from another node. Formally, let denote the set of messages receives in round , and let

   We require the following.

for all nodes and rounds , and

If node terminates in round , then .

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

6.1 Lower Bound for Centralized -Gossip in 1-Interval Connected Graphs

For this lower bound we assume that in each round , some central authority provides each node with a value 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 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 tokens for some positive constant .

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 nodes learn a new token, because connectivity guarantees that 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 for the -th token learned by each node: the first token each node learns comes at a cheap , and the last token learned costs dearly ( ). Formally, the potential of the system in round is given by

In the first round we have , because nodes know one token each. If the algorithm terminates in round then we must have , because all nodes must know all tokens. We construct an execution in which the potential increase is bounded by a constant in every round; this gives us an bound on the number of rounds required.

Theorem 6.1. *Any centralized algorithm for -gossip in 1-interval connected graphs requires rounds to complete in the worst case.*

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

Stage I: Adding the free edges. An edge is said to be *free* if and ; that is, if we connect and , neither node learns anything new. Let denote the set of free edges in round ; we add all of them to the graph. Let denote the connected components of the graph . Observe that any two nodes and in different components must send different values, otherwise we would clearly have and and and would be in the same component.

We choose representatives from each component arbitrarily. Our task now is to construct a connected subgraph over and pay only a constant cost. We assume that , otherwise we can connect the nodes arbitrarily for a constant cost. Let denote the number of tokens node does not know at the beginning of round .

Stage II: We split the nodes into two sets , according to the number of tokens they know, with nodes that know many tokens "on top": and consequently .

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 .

To that end, notice that : for all such that , either or , otherwise would be a free edge and would be in the same component; therefore each pair contributes at least one missing token to the sum. On the other hand, since each node in is missing at most tokens, it follows that . Putting the two facts together we obtain , and consequently also .

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 (see Fig. 1. That is, we are looking for a matching using only the edges and .

Since each top node is missing at most tokens, and each bottom node broadcasts a different value, for each top node there are at least edges in to choose from. But since we assume , ; thus, each top node can be connected to a different bottom node using -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

□

6.2 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

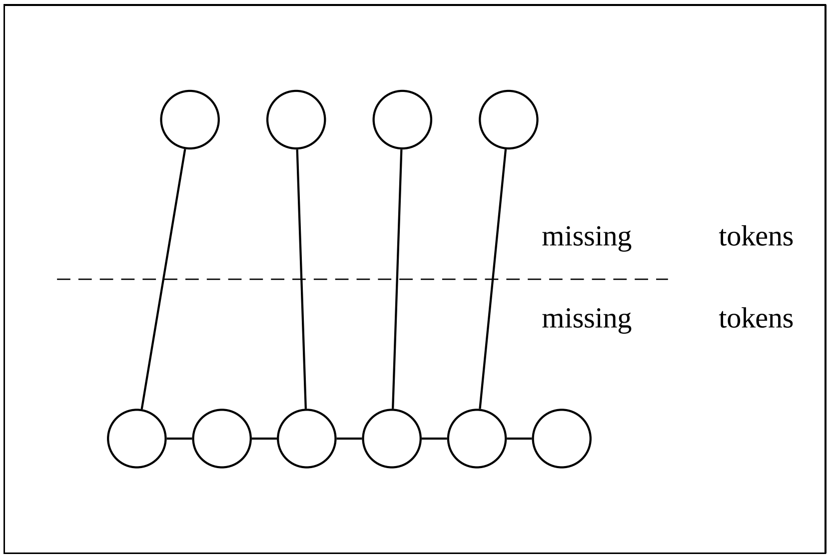


Figure 1: Illustration for the proof of the lower bound

A token-forwarding algorithm is said to be *knowledge-based* if it can be represented as a collection of functions , such that in every round , if is the sequence of coin-tosses for node up to round (inclusive), the distribution according to which node decides which token to broadcast is given by .

We say that two dynamic graphs and are *equal up to round* if and for all we have . Let denote the probability distribution for node in round . Knowledge-based algorithms have the following property.

Lemma 6.2. *Let be two dynamic graphs that are equal up to round , and let be an instance of gossip. If is a node such that , then for any round and string we have .*

Proof. Since and are equal up to round , the sequences and are equal, and in particular .

By definition, for all we have and ; therefore, for all . Consequently, for all , the sequences and are equal, and the claim follows. ☐

Theorem 6.3. *Any knowledge-based token-forwarding algorithm for -input gossip in -interval connected graphs over nodes requires rounds to succeed with probability at least . Further, if , then for sufficiently large , deterministic algorithms require rounds even when each node begins with at most one token.*

Proof. A lower bound of 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 .

Let be an knowledge-based token-forwarding algorithm for -gossip. We use the UID space as the token domain, and choose nodes : 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 starts with all tokens, and all other nodes start with a set . 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 in which and for all we have . In this starting state there are nodes that do not know each token . We abuse notation by using to denote the set of all tokens as well as the input assignment to each node .

Let . For a token , let denote the expected number of times token is broadcast by between rounds and (exclusive). We have

is broadcast in round

Thus, there are at least two tokens such that . Assume w.l.o.g. that . From Markov's inequality, node broadcasts less than times with probability at least in any execution fragment starting from round and ending before round , regardless of the dynamic graph we choose. The idea in the proof is to use as a buffer between the nodes that have already learned and those that have not; since broadcasts infrequently with high probability, in this manner we can limit the number of nodes that learn .

We divide the rounds between and into segments . The graph remains static during each segment, but changes between segments. For each segment we define two sets of nodes, and , where . The nodes in are "contaminated nodes" that might know token at the beginning of the segment; we connect them in a clique. The nodes in are "clean": initially, except for , these nodes do not know (some of them might learn during the segment). The only way the nodes in can learn is if broadcasts it. In the first segment is arranged in a line with at one end; in subsequent segments we "close" to form a ring. Initially and (recall that , 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 does not broadcast until the last round in the segment. In the last round of a quiet segment, broadcasts , and some nodes in the ring become contaminated. The first segment 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 to . We have to do this in a way that preserves -interval connectivity. Each active segment is triggered by broadcasting in the previous segment; if in some active segment does not broadcast , the next segment will be quiet.

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

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

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

( ) At the beginning of each quiet segment , node is the only node in the ring that knows token .

Let be some ordering of the nodes in (nodes that initially do not know). In each segment the nodes in will be some contiguous subset , where and for all . We place between and in the ring. Formally, the edges in any round where are given by

In the first segment, the edges are (we do not close the ring; this is to ensure that ( ) holds for the first active segment).

If is a quiet segment, then we define (and consequently); that is, the network does not change between and (except possibly for the closing of the ring after the first segment). However, if is an active session, then has some neighbors in the ring that knows , and they might spread to other nodes even when does not broadcast . We divide the nodes in into three subsets.

The *red nodes* comprise the nodes adjacent to on the contaminated side. The first of these (the ones closer to) may know at the beginning of the segment; the other may become contaminated if some of the first broadcast token . To be safe, we treat all red nodes as though they know by the end of the session.

The *yellow nodes* comprise the nodes adjacent to on the uncontaminated side. These nodes may learn during the segment, but only if broadcasts it.

The *green nodes* are all the other nodes in the ring. These nodes cannot become contaminated during the segment, because their distance from any node that knows is greater than .

Our cleanup between segments and consists of moving all the red nodes into . Formally, if , then we define and ; otherwise, if , then we define and . This satisfies ( ) and ( ): if does not broadcast during segment , then only the red nodes can know at the end, and since we removed them from the ring, at the beginning of no node knows except . The next segment will be quiet. Otherwise, if does broadcast during , then at the beginning of the next session (which is active) only the yellow nodes can know . These nodes then become red nodes in segment , and there are of them, as required.

The cleanup step preserves -interval connectivity: assume that (the other case is similar). Then the line exists throughout both segment and segment : in segment it exists as part of the ring, and in segment , after we moved the red nodes into the clique , the first part of the line exists in the clique and the second part exists in the ring. The nodes in are all connected to each other in both segments; thus, there is a static connected graph that persists throughout both segments , and in particular it exists in any rounds that start in . (Note that may be quiet, and in this case it can be shorter than rounds. But in this case it will be followed by an active segment which has exactly the same edges and lasts rounds.)

Notice that the number of uncontaminated nodes at the beginning of every active segment is at most less than in the previous active session. Therefore the total number of nodes that know by round is at most times the number of active sessions, and this in turn is bounded by times the number of rounds in which broadcasts . Since broadcasts less than times with probability at least , the algorithm is not finished by round with probability at least .

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 rounds, or
2. we reach a round in which some node has and for all we have .

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 nodes are their own UIDs, and the inputs to the other tokens are .

For deterministic algorithms the function representing node 's behavior must return a distribution in which one token has probability 1. We abuse notation slightly by using to denote this token.

We say that a process *fires in round* if when process receives as its input and hears nothing in the first rounds, it will stay silent in those rounds and then spontaneously broadcast its token in round . Formally, process fires in round if

1. For all we have , and
2. .

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

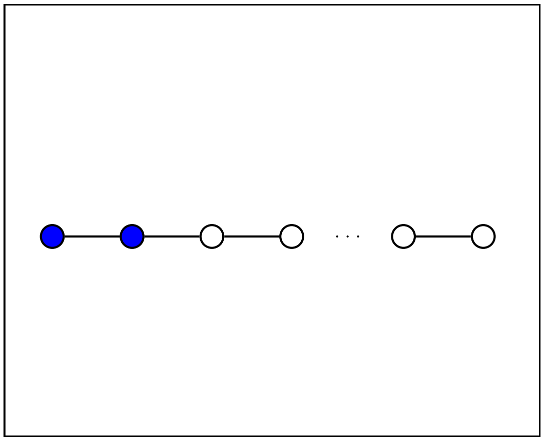
Since , there exist constants , such that for all we have . Let . We divide into two cases.

Case I. There exist that are all passive until round . In this case we construct the static clique over and let the algorithm run. During the first rounds, all nodes send only , and no node learns new tokens. Consequently all nodes have *in* , and the algorithm cannot terminate by round .

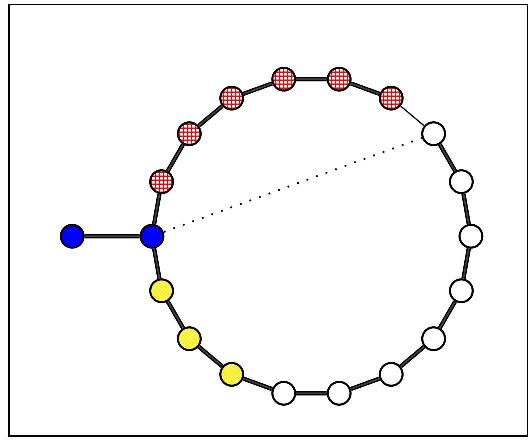
Case II. All but processes fire no later than round .

Since , by the pigeonhole principle there must exist a round such that at least processes fire in round . Let be such processes. We choose the instance where each node receives as input if , or if .

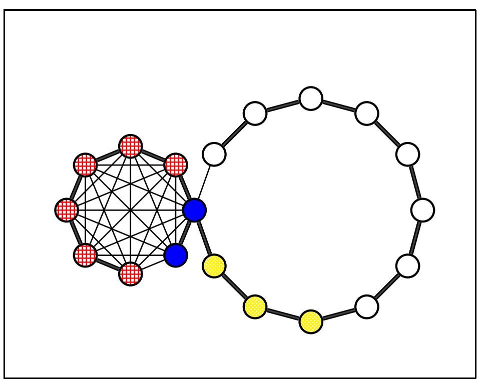
Let be the static star with at the center: , where for all . Because all nodes fire in round , when the algorithm is executed in , the network is silent until round . In round all nodes that have a token broadcast it. Following round we have , and for all , . This is the state from which we start the main body of the proof above. ☐



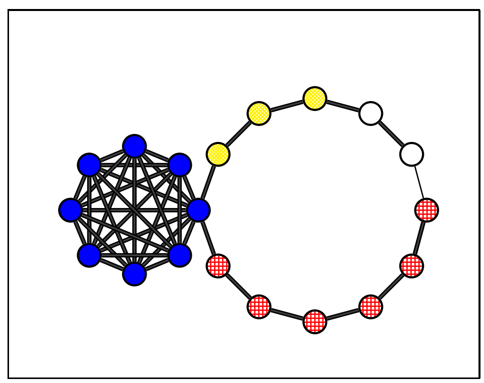
(a) The network at the beginning of the execution. Nodes that may know token are indicated in solid blue.



(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 -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 to . Double lines indicate stable edges along which -interval connectivity was preserved in the transition between the phases.



(d) If broadcast 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 's other side become yellow. Double lines indicate edges that will be stable through the next two phases.

Figure 2: Illustrations for the proof of the lower bound,

7 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 -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 Claim 5.1, 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 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] 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.

[15] 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.

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

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

[18] 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.

[19] 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.

[20] 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.

[21] 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.

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

[23] 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.

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

[25] 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.

[26] 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.

[27] 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.

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

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

[30] 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.

[31] 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.

[32] 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.

[33] 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.

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

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

[36] 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.