Chapter 10

Dynamic Networks

Many large-scale distributed systems and networks are dynamic. 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. In this chapter, we will study how to solve some basic tasks if the network is dynamic. Under what conditions is it possible to compute an accurate estimate of the size or some other property 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?

There are various reasons why networks can change over time and as a consequence, there also is a wide range of possible models for dynamic networks. Nodes might join or leave a distributed system. Some components or communication links may fail in different ways. Especially if the network devices are mobile, the connectivity between them can change. Dynamic changes can occur constantly or they might be infrequent enough so that the system can adapt to each change individually.

We will look at a synchronous dynamic network model in which the graph can change from round to round in a worst-case manner. To simplify things (and to make the problems we study well-defined), we assume that the set of nodes in the network is fixed and does not change. However, we will make almost no assumptions how the set of edges changes over time. We require some guarantees about the connectivity, apart from this, in each round, the communication graph is chosen in a worst-case manner by an adversary.

10.1 Synchronous Edge-Dynamic Networks

We model a synchronous dynamic network by a dynamic graph $G = (V, E)$, where V is a static set of nodes, and $E : \mathbb{N}_0 \to \binom{V}{2}$ is a function mapping a round number $r \in \mathbb{N}_0$ to a set of undirected edges $E(r)$. Here $\binom{V}{2} := \{\{u, v\} \mid u, v \in V\}$ is the set of all possible undirected edges over V.

Definition 10.1 (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*.*

For simplicity, we restrict to deterministic algorithms. Nodes communicate with each other using *anonymous broadcast*. At the beginning of round r, each node u decides what message to broadcast based on its internal state; at the same time (and independently), the adversary chooses a set $E(r)$ of edges for the round. As in standard synchronous message passing, all nodes v for which $\{u, v\} \in E(r)$ receive the message broadcast by node u in round r and each node can perform arbitrary local computations upon receiving the messages from its neighbors. We assume that all nodes in the network have a unique identifier (ID). In most cases, we will assume that messages are restricted to $O(\log n)$ bits. In these cases, we assume that node IDs can be represented using $O(\log n)$ bits, so that a constant number of node IDs and some additional information can be transmitted in a single message. We refer to the special case where all nodes are woken up at once as *synchronous start* and to the general case as *asynchronous start*.

We assume that each node in the network starts an execution of the protocol in an initial state which contains its own ID and its input. Additionally, nodes know nothing about the network, and initially cannot distinguish it from any other network.

10.2 Problem Definitions

In the context of this chapter, we study the following problems.

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

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

***k*-token dissemination.** An instance of k-token dissemination is a pair (V, I), where $I : V \to \mathcal{P}(\mathcal{T})$ assigns a set of tokens from some domain $\mathcal{T}$ to each node, and $|\bigcup_{u \in V} I(v)| = k$. An algorithm solves k-token dissemination if for all instances (V, I), when the algorithm is executed in any dynamic graph $G = (V, E)$, all nodes eventually terminate and output $\bigcup_{u \in V} I(u)$. We assume that each token in the nodes' input is represented using $O(\log n)$ bits. Nodes may or may not know k, depending on the context. Of particular interest is *all-to-all token dissemination*, a special case where $k = n$ and each node initially knows exactly one token, i.e., $|I(u)| = 1$ for all nodes u.

***k*-committee election.** As an useful step towards solving counting and token dissemination, we consider a 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 terminate and output a committee ID such that the two conditions are satisfied.

10.3 Basic Information Dissemination

To start, let us study how a single piece of information is propagated through a dynamic network. We assume that we have a dynamic network graph G with n nodes such that G is always connected (G is 1-interval connected as defined in Definition 10.1). Further assume that there is a single piece of information (token), which is initially known by a single node.

Theorem 10.2. *Assume that there is a single token in the network. Further assume that at time* 0 *at least one node knows the token and that once they know the token, all nodes broadcast it in every round. In a* 1*-interval connected graph* $G = (V, E)$ *with* n *nodes, after* $r \leq n - 1$ *rounds, at least* $r + 1$ *nodes know the token. Hence, in particular after* $n - 1$ *rounds, all nodes know the token.*

Proof. We can proof the theorem by induction on r. Let $T(r)$ be the set of nodes that know the token after r rounds. We need to show that for all $r \geq 0$, $|T(r)| \geq \min\{r + 1, n\}$. Because we assume that at time 0 at least one node knows the token, clearly, $|T(0)| \geq 1$. For the induction step, assume that after r rounds, $|T(r)| \geq \min\{r + 1, n\}$. If $T(r) = V$, we have $|T(r + 1)| \geq |T(r)| = n$ and we are done. Otherwise, we have $V \setminus T(r) \neq \emptyset$. Therefore, by the 1-interval connectivity assumption, there must be two nodes $u \in T(r)$ and $v \in V \setminus T(r)$ such that $\{u, v\} \in E(r + 1)$. Hence, in round $r + 1$, node v gets the token an therefore $|T(r + 1)| \geq |T(r)| + 1 \geq \min\{r + 2, n\}$. □

Remarks:

- Note that Theorem 10.2 only shows that after $n-1$ rounds all nodes know the token. If the nodes do not know n or an upper bound on n, they do not know if all nodes know the token.
- We can apply the above techniques also if 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 *know* when they know the smallest or biggest token.

The next theorem shows that essentially, for the general asynchronous start case, 1-interval connectivity does not suffice to obtain anything better than what is stated by the above theorem. If nodes do not know n or an upper bound on n initially, they cannot find n.

Theorem 10.3. *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 static line of length n from a dynamically changing line of length n'.

Given a sequence $A = a_1 \circ \ldots \circ a_m$, let $\text{shift}(A, r)$ 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 $B(r) = \text{shift}(n \circ \ldots \circ (n'-1), r)$ 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. □

Remark:

- The above impossibility result extends to all problems introduced in Section 10.2 as long as we do not assume that the nodes know n or an upper bound on n.

In light of the impossibility result of Theorem 10.3, let us now first consider the synchronous start case where all nodes start the protocol at time 0 (with round 1). We first look at the case where there is no bound on the message size and describe a simple linear-time protocol for counting (and token dissemination). The protocol is extremely simple, but it demonstrates some of the ideas used in some of the 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 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 $|A| \leq r$.

```
A ← {self};
for r = 1, 2, ... do
    broadcast A;
    receive B_1, ..., B_s from neighbors;
    A ← A ∪ B_1 ∪ ... ∪ B_s;
    if |A| ≤ r then terminate and output |A|;
end
```

Algorithm 3: Counting in linear time using large messages

Before analyzing Algorithm 3, let us fix some notation that will help to argue about the algorithms we will study. If x is a variable of an algorithm, let $x_u(r)$ be the value of the variable x at node u after r rounds (immediately before the broadcast operation of round $r+1$). For instance in Algorithm 3, $A_u(r)$ denotes the set of IDs of node u at the end of the r^{th} iteration of the for-loop.

Lemma 10.4. *Assume that we are given an 1-interval connected graph $G = (V, E)$ and that all nodes in V execute Algorithm 3. If all nodes together start at time 0, we have $|A_u(r)| \geq r + 1$ for all $u \in V$ and $r < n$.*

Proof. We prove the lemma by induction on r. We clearly have $|A_u(0)| = 1$ for all u because initially each node includes its own ID in A. Hence, the lemma is true for $r = 0$.

For the induction step, assume that the claim of the lemma is true for some given $r < n - 1$ for all dynamic graphs G. Let $A'_u(r+1)$ be the set of identifiers known by node u if all nodes start the protocol at time 1 (instead of 0) and run it for r rounds. By the induction hypothesis, we have $|A'_u(r+1)| \geq r+1$. If the algorithm is started at time 0 instead of time 1, the set of identifiers in $A_u(r+1)$ is exactly the union of all the identifiers known by the nodes in $A'_u(r+1)$ after the first round (at time 1). This includes all the nodes in $A'_u(r+1)$ as well as their neighbors in the first round. If $|A'_u(r+1)| \geq r+2$, we also have $|A_u(r+1)| \geq r+2$ and we are done. Otherwise, by 1-interval connectivity, there must at least be one node $v \in V \setminus A'_u(r+1)$ for which there is an edge to a node in $A'_u(r+1)$ in round 1. We therefore have $|A_u(r+1)| \geq |A'_u(r+1)| + 1 \geq r + 2$. □

Theorem 10.5. *In an 1-interval connected graph G, Algorithm 3 terminates at all nodes after n rounds and output n.*

Proof. Follows directly from Lemma 10.4. For all nodes u, $|A_u(r)| \geq r + 1 > r$ for all $r < n$ and $|A_u(n)| = |A_u(n-1)| = n$. □

Lemma 10.6. *Assume that we are given a 2-interval connected graph $G = (V, E)$ and that all nodes in V execute Algorithm 3. If node u is waken up and starts the algorithm at time t, it holds that have $|A_u(t + 2r)| \geq r + 1$ for all $0 \leq r < n$.*

Proof. The proof follows along the same lines as the proof of Lemma 10.4 (see exercises). □

Remarks:

- Because we did not bound the maximal message size and because every node receives information (an identifier) from each other node, Algorithm 3 can be used to solve all the problems defined in Section 10.2. For the token dissemination problem, the nodes also need to attach a list of all known tokens to all messages

- As a consequence of Theorem 10.3, 1-interval connectivity does not suffice to compute the number of nodes n in a dynamic network if nodes start asynchronously. It turns out that in this case, we need a slightly stronger connectivity assumption. If the network is 2-interval connected instead of 1-interval connected, up to a constant factor in the time complexity, the above results can also be obtained in the asynchronous start case (see exercises).

- For the remainder of the chapter, we will only consider the simpler synchronous start case. For $T \geq 2$, all discussed results that hold for T-interval connected networks with synchronous start also hold for asynchronous start with the same asymptotic bounds.

10.4 Small Messages

We now switch to the more interesting (and more realistic) case where in each round, each node can only broadcast a message of $O(\log n)$ bits. We will first show how to use k-committee election to solve counting. We first describe how to obtain a good upper bound on n. We will then see that the same algorithm can also be used to find n exactly and to solve token dissemination.

10.4.1 k-Verification

The 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.

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

Next, suppose $k < n$. We can show that after the ith round of the protocol, at least i nodes in each committee have $x = 0$. In any round of the protocol, consider a cut between the nodes that belong to a particular committee and still have $x = 1$, and the rest of the nodes, which either belong to a different committee or have $x = 0$. From 1-interval connectivity, there is an edge in the cut, and some node u in the committee that still has $x_u = 1$ hears either a different committee ID or $\perp$. Node u then sets $x_u \leftarrow 0$, and the number of nodes in the committee that still have $x = 1$ decreases by at least one. Since each committee initially contains at most k nodes, after k rounds all nodes in all committees have $x = 0$, and all output 0. □

10.4.2 k-Committee Election

We can solve k-committee in $O(k^2)$ rounds as follows. Each node u stores two local variables, $committee_u$ and $leader_u$. A node that has not yet joined a

```
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 ← ⊥;
    end
    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};
    end
    // 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 ← ⊥
    end
    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)
    end
    // Join the leader's committee, if invited
    if invitation = (leader, self) then
        committee = leader;
    end
end
if committee = ⊥ then
    committee ← self;
end
```

Algorithm 4: k-committee in always-connected graphs

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 the 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 $committee_u = u$. The details are given in Algorithm 4.

Lemma 10.8. *Algorithm 4 solves the k-committee problem in $O(k^2)$ rounds in 1-interval connected networks.*

Proof. The time complexity is immediate. To prove correctness, 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 all-to-all token dissemination 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.

10.5 More Stable Graphs

```
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}
        end
        receive t_1, ..., t_s from neighbors;
        A ← A ∪ {t_1, ..., t_s}
    end
    S ← ∅
end
return A
```

Procedure disseminate(A, T, k)

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. Basically, because we are guaranteed that some edges and paths persist for T rounds, it suffices to send a particular ID or token only once in T rounds to guarantee progress. Other rounds can then be used for different tokens. For convenience we assume that the graph is $2T$-interval connected for some $T \geq 1$.

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

The execution of procedure `disseminate` is divided into $\lceil k/T \rceil$ phases, each consisting of $2T$ rounds. During each phase, each node maintains the set A of tokens it has already learned and a set S of tokens it has already broadcast in the current phase (initially empty). In each round of the phase, the node

broadcasts the smallest token it has not yet broadcast in the current phase, then adds that token to S.

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 $\text{dist}_i(u, v)$ 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 $\text{tdist}_i(u, t)$ to be the distance of u from the nearest node in G'_i that knows t at the beginning of phase i:

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

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 $K^i_t(r) := K_t(2T \cdot i + r)$. Correctness hinges on the following property.

Lemma 10.9. *For any node $u \in V$, token $t \in \bigcup_{v \in V} A_v(0)$, and round r such that $\text{tdist}_i(u, t) \leq r \leq 2T$, either $t \in S^i_u(r + 1)$ or $S_u(r + 1)$ includes at least $(r - \text{tdist}_i(u, t))$ 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 $r \geq \text{tdist}_i(u, t)$. If $r = \text{tdist}_i(u, t)$, then $r - \text{tdist}_i(u, t) = 0$ and the claim holds trivially. Thus, suppose that $r > \text{tdist}_i(u, t)$. Hence, $r - 1 \geq \text{tdist}_i(u, t)$, and the induction hypothesis applies: either $t \in S^i_u(r)$ or $S^i_u(r)$ includes at least $(r - 1 - \text{tdist}_i(u, t))$ tokens that are smaller than t. In the first case we are done, since $S^i_u(r) \subseteq S^i_u(r+1)$; thus, assume that $t \notin S^i_u(r)$, and $S^i_u(r)$ includes at least $(r - 1 - \text{tdist}_i(u, t))$ tokens smaller than t. However, if $S^i_u(r)$ includes at least $(r - \text{tdist}_i(u, t))$ tokens smaller than t, then so does $S^i_u(r + 1)$, and the claim is again satisfied; thus we assume that $S^i_u(r)$ includes *exactly* $(r - 1 - \text{tdist}_i(u, t))$ tokens smaller than t.

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

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

Next suppose that $\text{tdist}_i(u, t) > 0$. Let $x \in K^i_t(0)$ be a node such that $\text{dist}_i(u, x) = \text{tdist}(u, t)$ (such a node must exist from the definition of $\text{tdist}_i(u, t)$), and let v be a neighbor of u along the path from u to x in G_i, such that $\text{dist}_i(v, x) = \text{dist}_i(u, x) - 1 < r$. From the induction hypothesis, either $t \in S^i_v(r)$ or $S^i_v(r)$ includes at least $(r - 1 - \text{tdist}_i(v, t)) = (r - \text{tdist}_i(u, t))$ 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 $S^i_v(r) \subseteq A^i_u(r)$. Finally, because we assumed that $S^i_u(r)$ contains exactly $(r - 1 - \text{tdist}_i(u, t))$ 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. □

Using Lemma 10.9 we can show: correct.

Lemma 10.10. *If $k \geq n$, at the end of procedure* `disseminate` *the set A_u of each node u contains the T smallest tokens.*

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

From Lemma 10.9, 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 $\mathbb{N}_i^T(t)$ 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 new nodes learn t, until all the nodes know t. Since there are no more than k nodes and we have $\lceil k/T \rceil$ phases, at the end of the last phase all nodes know t. □

To solve counting and token dissemination with up to n tokens, we use Procedure `disseminate` to speed up the k-committee election protocol from Algorithm 4. Instead of inviting one node in each cycle, we can use `disseminate` to have the leader learn the IDs 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 10.11. *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 $O(n+n^2/T)$-round protocols for counting all-to-all token dissemination.*

Remarks:

- The same result can also be achieved for the asynchronous start case, as long as $T \geq 2$.
- The described algorithm is based on the assumptions that all nodes know T (or that they have a common lower bound on T). At the cost of a log-factor, it is possible to drop this assumption and adapt to the actual interval-connectivity T.
- It is not known whether the bound of Theorem 10.11 is tight. It can be shown that it is tight for a restricted class of protocols (see exercises).
- If we make additional assumptions about the stable subgraphs that are guaranteed for intervals of length T, the bound in Theorem 10.11 can be improved. E.g., if intervals of length T induce a stable k-vertex connected subgraph, the complexity can be improved to $O(n + n^2/(kT))$.