Distributed Computation in Wireless and Dynamic Networks

by

Rotem Oshman

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

Doctor of Philosophy

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

September 2012

Author ....
Department of Electrical Engineering and Computer Science
July 10, 2012

Certified by ....
Nancy A. Lynch
Professor of Electrical Engineering and Computer Science
Thesis Supervisor

Accepted by ....
Professor Leslie A. Kolodziejski
Chairman, Department Committee on Graduate Students

Distributed Computation in Wireless and Dynamic Networks

by
Rotem Oshman

Submitted to the Department of Electrical Engineering and Computer Science
on July 10, 2012, in partial fulfillment of the
requirements for the degree of
Doctor of Philosophy

Abstract

Today's wireless networks tend to be centralized: they are organized around a fixed central backbone such as a network of cellular towers or wireless access points. However, as mobile computing devices continue to shrink in size and in cost, we are reaching the point where large-scale ad-hoc wireless networks, composed of swarms of cheap devices or sensors, are becoming feasible. In this thesis we study the theoretical computation power of such networks, and ask what tasks are they capable of carrying out, how long does solving particular tasks take, and what is the effect of the unpredictable network topology on the network's computation power.

In the first part of the thesis we introduce an abstract model for dynamic networks. In contrast to much of the literature on mobile and ad-hoc networks, our model makes fairly minimalistic assumptions; it allows the network topology to change arbitrarily from round to round, as long as in each round the communication graph is connected. We show that even in this weak model, global computation is still possible, and any function of the nodes' initial inputs can be computed efficiently. Also, using tools from the field of epistemic logic, we analyze information flow in dynamic networks, and study the time required to achieve various notions of coordination.

In the second part of the thesis we restrict attention to static networks, which retain an important feature of wireless networks: they are potentially *asymmetric*. We show that in this setting, classical data aggregation tasks become much harder, and we develop both upper and lower bounds on computing various classes of functions. Our main tool in this part of the thesis is communication complexity: we use existing lower bounds in two-player communication complexity, and also introduce a new problem, *task allocation*, and study its communication complexity in the two-player and multi-player settings.

Thesis Supervisor: Nancy A. Lynch
Title: Professor of Electrical Engineering and Computer Science

Acknowledgments

First and foremost, I wish to thank Nancy Lynch for her advice, support, and mentorship, and for passing on to me some of her boundless energy and devotion to research. Working with Nancy and being part of her research group has been an inspiring and enriching experience, and I am indebted to her for her amazing dedication and patience during the process of writing this thesis. It has been my privilege to be Nancy's student, and I am truly grateful for the experience.

I would also like to thank my collaborators, Andrew Drucker, Fabian Kuhn, Christoph Lenzen, Thomas Locher, Thomas Moscibroda, and Yoram Moses, who worked with me on the results in this thesis and elsewhere. It has been a pleasure to work with you.

To Andrew Drucker, with whom I have shared an office these past four years: thank you for many fascinating discussions, about both research and not-research. Our conversations have broadened my view of theoretical computer science and enriched my experience as a graduate student.

Thanks to Hagit Attiya and Nir Shavit for their advice and support throughout my graduate studies at the Technion and at MIT.

A special thanks to Fabian Kuhn, who worked with me on all the results in this thesis. I was lucky to begin my graduate studies at MIT in the same year that Fabian was a post-doctoral associate in the Theory of Distributed Systems Group, and our collaboration throughout my graduate studies has been both fruitful and fun. I thank Fabian for hosting me in Lugano on many occasions. It has been my great pleasure and privilege to work with you.

To my wonderful parents, my beautiful siblings, and the entire clan.
Your support has meant the world to me.

Contents

1 Introduction **13**
1.1 Modeling Wireless and Dynamic Networks . . . 13
1.2 A Hierarchy of Distributed Tasks . . . 14
1.3 Overview of the Results . . . 16
1.3.1 Token Dissemination . . . 16
1.3.2 Counting and Approximate Counting . . . 16
1.3.3 Minimum and Similar "Easy" Functions . . . 17
1.3.4 Consensus and Coordinated Consensus . . . 18
1.3.5 Finding Spanning Trees and Solving Task Allocation . . . 18
1.4 Related Work . . . 19
1.4.1 Models for Dynamic Networks . . . 20
1.4.2 Computation in Wireless Networks . . . 23
1.5 Organization of the Results . . . 26
1.6 Definitions and Notation . . . 27

I Distributed Computation in Dynamic Networks **29**

2 The Dynamic Graph Model **31**
2.1 Dynamic Graphs: an Abstract Model of Dynamic Networks . . . 31
2.2 Single-Shot Computation Tasks . . . 33
2.3 Connectivity, Expansion and Information Flow in Dynamic Networks 38
2.4 Epistemic Logic and Information Flow . . . 46

3 Counting and Information Dissemination in Dynamic Networks **49**
3.1 Problem Definitions and Basic Properties . . . 50
3.2 Counting and Information Dissemination using Large Messages . . . 55
3.3 Counting Through k-Committee Election . . . 56
3.4 Deterministic Counting and All-to-All Token Dissemination . . . 59
3.4.1 k-Committee in 1-Interval Connected Graphs . . . 59
3.4.2 All-to-All Token Dissemination in ∞-Interval Connected Graphs 63
3.4.3 Counting and Token Dissemination in T-Interval Connected Graphs . . . 66
3.5 Other Variations on the Model . . . 71
3.5.1 Graphs with Unknown Interval Connectivity and Vertex Growth 73

3.5.2 Coping with Asynchronous Wakeup 75
3.5.3 Counting and Token Dissemination with Beeps 78
3.6 Randomized Approximate Counting 79
3.7 Lower Bound on Token Dissemination by Token-Forwarding 86
3.8 Summary and Open Problems 96

4 Consensus and Coordination in Dynamic Networks 99
4.1 Epistemic Logic in Distributed Computing 100
4.2 Consensus and Causality 104
4.3 Simultaneous Tasks and Common Knowledge 107
4.3.1 Simultaneous Tasks 107
4.3.2 Common Knowledge in Dynamic Graphs 108
4.3.3 Application to Simultaneous Consensus 116
4.4 Δ-Coordinated Consensus 120
4.5 Lower Bounds on Δ-Coordinated Consensus 127
4.5.1 General Setup 127
4.5.2 Lower Bound For Static Paths 129
4.5.3 Lower Bound for Graphs of Specific Dynamic Diameter 134

II The Communication Complexity of Distributed Computation in Directed Networks 138

5 Directed Networks and Two-Player Communication Games 139
5.1 Directed Broadcast Networks 139
5.2 Two-Player Communication Complexity 140
5.2.1 Two-Player Communication Games 140
5.2.2 Lower Bounds in Communication Complexity 141
5.2.3 Applying Communication Complexity Lower Bounds in Distributed Computing 145

6 Data Aggregation in Directed Networks 147
6.1 Approximate and Exact Counting 147
6.1.1 Lower Bound on Approximate Counting 147
6.1.2 Deterministic and Exact Counting 150
6.1.3 A Nearly-Matching Upper Bound 150
6.1.4 Lower Bound on Finding a Rooted Spanning Tree 152
6.2 Lower Bounds on Sensitive Functions 152
6.2.1 Lower Bounds on the HF_m Task 153
6.2.2 A Reduction from Set Disjointness to HF_m 154
6.2.3 Discussion and Extensions 157
6.3 A Nearly-Optimal Algorithm for HF_n 159
6.3.1 Discussion and High-Level Sketch of the Algorithm 159
6.3.2 Token Selection Schemes 160
6.3.3 Solving HF_n using Token Selection 163

6.3.4 Simultaneous HF_n 170

7 Distributed Task Allocation in Directed Networks 175
7.1 Problem Statement 177
7.2 Two-Player Lower Bound for Task Allocation 178
7.3 Lower Bound on Rooted Spanning Tree 181
7.4 Multiparty Complexity in the Shared Blackboard Model 183
7.5 Arbitrary Networks 192
7.6 An Algorithm for Constructing Rooted Spanning Trees 194
7.7 Discussion and Open Problems 195

8 Conclusion 197
8.1 Summary of the Results 197
8.1.1 Information Dissemination 197
8.1.2 Counting and Computing Functions of the Input 197
8.1.3 Coordinated Consensus 198
8.1.4 Task Allocation 198
8.2 Discussion and Concluding Remarks 198
8.2.1 Designing Algorithms for Ad-Hoc Wireless Networks 198
8.2.2 Lower Bound Techniques 200
8.3 Open Problems 201
8.3.1 Direct Extensions of the Results in this Thesis 201
8.3.2 Designing Better Algorithms for Ad-Hoc Networks 203
8.3.3 Understanding the Information-Theoretic Principles Underlying Distributed Computation 205

Index 209

Chapter 1

Introduction

1.1 Modeling Wireless and Dynamic Networks

Sequential models of computation, such as Turing machines, are robust to changes in the assumptions of the model.[1] It does not matter if a Turing machine has one working tape or two, or even infinitely many; it does not matter whether its alphabet is binary or ternary, or whether it is deterministic or can use random coins — the computation power of the Turing machine is unaffected. In contrast, models of *distributed* systems are notoriously fragile and non-robust. Tasks such as *fault-tolerant consensus*, which can be solved using randomness, can become impossible in deterministic models; *Byzantine consensus* is possible to solve if up to $\lceil n/3 \rceil - 1$ nodes are faulty, but not if $\lceil n/3 \rceil$ nodes are faulty. Consequently the distributed computing community, perhaps more than any other sub-field in theoretical computer science, is interested in the minute details of the modeling assumptions, and the effect that they can have on computation power and efficiency.

In this thesis we study wireless networks, in particular dynamic ones, and their computation power. Wireless networks are inherently difficult to model: unlike traditional wired networks, wireless networks do not come equipped with a built-in network graph, and modeling them using graph-theoretic formalisms can be non-trivial. Instead of pairwise communication over wires, which is easy to represent as an edge in the network graph, wireless nodes broadcast their messages on the wireless medium, to be received by whichever nodes happen to be nearby and receive the signal with sufficiently high signal-to-noise ratio. Interference and noise render wireless communication difficult to pin down in a clean way; traditional assumptions such as *bidirectional communication* and *reliability* no longer apply. These concerns arise even in static networks, and adding mobility into the picture only complicates matters further.

Much of the literature in the theory of wireless communication attempts to model wireless networks using a *geographical model*, which assumes that the wireless nodes

[1]This assertion is often referred to as the *Church-Turing thesis*, although there is considerable controversy over the exact nature of the theses formulated by Church and Turing in their writings, and indeed whether either of them intended anything near the modern interpretation of the Church-Turing thesis [121].

are embedded in the two-dimensional (or three dimensional) plane, and two nodes can communicate with each other only if their locations satisfy some geometric condition. For example, the popular *unit disk graph model* [31, 87, 99, 25] assumes that two nodes can reliably communicate if and only if they are within a fixed distance from each other. However, it is recognized [19, 99] that such simplistic models, while amenable to theoretical analysis, do not always represent the reality of wireless communication. We survey existing models in the literature in Section 1.4.1.

The approach we take in this thesis is different. Instead of attempting to model the low-level details of physical wireless communication, we model wireless networks at a higher level, above the medium-access (MAC) layer. We assume that some MAC protocol is responsible for message delivery and all that it entails; this protocol interfaces with higher levels in the stack by means of a synchronous (round-based) interface, which guarantees that in each round, the graph induced by all the messages successfully delivered to their destinations is strongly-connected. Beyond this we make few or no assumptions. In particular, in Part I of the thesis, we do not assume a static topology or reliable message delivery, and allow the communication graph induced by the delivered messages to change arbitrarily from round to round. In Part II of the thesis we do assume that the network is static and communication is reliable — as occurs, for example, when nodes initially establish some time-division scheme and adhere to it thereafter. In both cases, we strive to avoid many restrictive assumptions often made in the literature, such as:

- A fixed set of participants, with UIDs labeled $1, \ldots, n$; or, somewhat weaker,
- A fixed *number* of participants with unknown UIDs;
- A known diameter, maximum degree, or even entire topology of the communication graph;
- Bi-directional communication;
- Information regarding the node's neighbors in the network (*neighbor discovery*);
- Predictable mobility patterns for the nodes.

Where possible, we study the effects of these assumptions on the computation power of the model and the efficiency of algorithms running in it. Our aim is to establish upper bounds (i.e., algorithms) using as few assumptions as possible, and lower bounds under as strong a model as possible, in order to point out which assumptions are or are not relevant to a particular problem. Our model is much more abstract and general than many existing models in the literature, as we discuss in Section 1.4.1.

1.2 A Hierarchy of Distributed Tasks

In this thesis we will focus mainly on the following distributed tasks, listed here in decreasing order of their relative difficulty:

1. *Information dissemination*, where nodes must disseminate up to n pieces of information throughout the network.

2. *Counting* and *approximate counting*, where nodes compute or approximate the size n of the network.

3. *Computing duplication-insensitive functions*, such as the minimum or maximum input value.

4. *Consensus*, where each node receives a private binary input, and the goal is for all nodes to agree on the same output, which must also be the input to one of the nodes.

We will show in this thesis that an oracle for solving each task in the hierarchy allows us to solve all lower-level tasks (see Section 2.2 for our definition of a *reduction* in the context of a distributed dynamic network); this is mostly obvious, but the connection between counting and computing "simple" functions is perhaps less obvious (it will be discussed in Section 3.1). It is therefore interesting to ask whether the hierarchy is *strict*, or whether in fact all these problems are equally hard. The answer depends on the assumptions used, and we show several surprising results; for example,

- If nodes do not know the size of the network in advance, solving consensus is as hard as computing the minimum input, even though consensus does not specify a deterministic output value, while the task of computing a minimum does.

- Moreover, if we want to solve consensus and have all nodes terminate *simultaneously*, this task is as hard as computing an upper bound on the size of the network, and it requires at least n rounds in *every single execution*, even very well behaved-ones.

- Computing the size of the network requires at least two nodes in the network to exchange a total of $\Omega(n)$ bits with each other. Even if the network is static and the diameter of the network is known in advance to be 2, this task requires $\Omega(n/B)$ rounds, where B is an upper bound on the number of bits in each message.

- In a similar vein, if the diameter of the network is *not* known in advance, then solving consensus requires $\Omega(\sqrt{n/B})$ rounds, even when in practice the diameter of the network is 2.

These results are all negative in nature: they show that a problem is unconditionally hard, or that one problem is at least as hard as another. We also give many positive results; e.g.,

- Even if nodes know nothing about the network in advance except for their own unique identifier, and the network topology can change arbitrarily from round to round, it is still possible to deterministically compute any function of the nodes' initial inputs.

- Randomization and approximation allow us to circumvent some of the bounds above. For example, if we are willing to settle for a high-probability constant approximation for the size of a static network, we can compute it in $\tilde{O}(D)$ rounds when the diameter is D.

1.3 Overview of the Results

We now briefly describe several results that form the core of this thesis.[2]

In the sequel, we let n denote the number of nodes in the network, and D its diameter.

1.3.1 Token Dissemination

In Chapter 3, we show that in highly-dynamic networks it is possible to solve the problem of *all-to-all token dissemination*, where each node receives some input token and must collect all tokens from all n nodes. This is possible in $O(n^2)$ rounds using messages of size $O(\log n)$, even when n is not known in advance, and even when nodes have no *a priori* information about the network other than their own unique identifier. The algorithm is quite pessimistic, as it assumes that the network topology is generated on-the-fly by a worst-case adversary, and may change completely from round to round. We also show that when the network enjoys some stability, it is possible to improve upon the running time: if there is some underlying connected subgraph that is present throughout the execution, then n pieces of information can be exchanged in $O(n)$ rounds. We then combine ideas from the $O(n^2)$-round "pessimistic" algorithm and the $O(n)$-round "optimistic" algorithm to obtain an algorithm that makes an intermediate assumption, namely that any T consecutive rounds contain some persistent connected subgraph. This algorithm requires $O(n + n^2/T)$ rounds, and can be generalized to the case where T, the "amount of stability", is not known in advance. We conclude Chapter 3 by drawing several generalizations and extensions of the results in the chapter, and then describing an $\Omega(n + n^2/T)$-round lower bound on a specific type of information dissemination algorithm (which includes all the algorithms we give in this thesis). Most of the results in Chapter 3 are deterministic in nature, but our lower bound also applies to randomized algorithms.

1.3.2 Counting and Approximate Counting

When the size of the network is not known in advance, the problem of determining this parameter is tightly coupled to the problem of all-to-all information dissemination: in order for a node to *know* when it has collected all the tokens from the other nodes, it must implicitly figure out *how many* other nodes there are. Thus, the information-dissemination algorithms that we present in Chapter 3 also compute the size of the network as they go along, in order to determine when it is safe to halt. In Chapter 3

[2]The results are grouped here by problem, not by chapter. Please see Section 1.5 for a chapter-by-chapter roadmap.

we also give a randomized approximate-counting algorithm that requires nearly-linear time, improving upon the $O(n^2)$-round running time of the deterministic information dissemination/counting algorithm from that chapter.

We return to study counting in the second part of the thesis, where we consider static directed networks. We now study the hardness of counting and related tasks through the lens of *communication complexity*, which will allow us to investigate *how many bits must be exchanged* to solve these tasks. In Chapter 6, we show that even in strongly-connected networks of diameter 2, counting requires $\Omega(n/B)$ rounds, where B is an upper bound on the message size. This is a somewhat suprising result, as it is well-known that in *undirected* networks of diameter 2, the same task only requires $O(1)$ rounds (regardless of message size [110]). This lower bound is tight if the diameter of the network is known in advance to be 2, but it degrades to a constant if we allow a constant approximation instead of an exact count. However, if the diameter is *not* known in advance, we derive an $\Omega(\sqrt{n/B})$-round lower bound on computing even an approximate count.

1.3.3 Minimum and Similar "Easy" Functions

Througout the thesis, we will treat the task of computing the minimum input to the nodes as representative of a large class of "easy" functions, which are not sensitive to duplication of inputs (unlike, for example, counting). To compute the minimum input, all nodes have to do is send the smallest value they have heard so far, "wait long enough" to ensure they have heard the global minimum, and then halt. How long is long enough? We encapsulate this question as a separate task, which we call *hearing from everyone* (and denote HF_n): informally, hearing from everyone requires each node u to halt at some point, but only after node u has "heard from" each node v in the network by means of a chain of messages starting at v and ending at u.[3]

It is easy to show that computing the minimum input *requires* hearing from everyone. On the other hand, if we have an algorithm for solving the HF_n task, we can also easily compute the minimum, by simply appending the minimum value heard so far to each message of the HF_n algorithm. Other duplication-insensitive functions behave similarly; thus, the HF_n task is a good proxy for this class, and any upper or lower bound we establish on HF_n will carry over to all duplication-insensitive functions. (We make this argument more formally in Section 3.1, where we prove that HF_n is *complete* for the class of duplication-insensitive functions.)

Because HF_n is such a fundamental task, many of the algorithms we give in this thesis implicitly solve it, even though they may be targeted at higher-level tasks. For example, to solve information-dissemination or counting, one must also solve HF_n; all the algorithms given in Chapter 3 therefore solve the HF_n task. However, in Chapter 6 we devote some time to studying the HF_n task for its own sake. We show that if the diameter of the network is not known in advance, but in practice *happens* to be 2, solving HF_n requires $\Omega(\sqrt{n/B})$ rounds; in other words, HF_n algorithms cannot adapt to conditions "on the ground" and cannot exploit a small network diameter when it

[3]This notion is *Lamport's causal order*; see Section 2.3 for the definition as we use it here.

arises in practice. On the positive side, we give a nearly-matching randomized algorithm for HF_n, which of course also allows us to compute any duplication-insensitive function.

1.3.4 Consensus and Coordinated Consensus

Consensus is one of the most well-known problems in distributed computing; it is useful for modeling many different situations that arise in practice, from deciding which version of a file to commit to having all members of a swarm agree on the direction they should move in. We study consensus, and several variants of consensus that also include timing constraints, in Chapter 4.

Chapter 4 stands apart from the rest of the thesis in that it considers *full-information algorithms*, where nodes send their entire history in each message (leading, unsurprisingly, to very large messages). Full-information algorithms are interesting because they extract as much information as possible from the execution: because nodes send each other everything they have observed about the execution, each node learns everything it can possibly learn about the execution as quickly as it possible. Thus, lower bounds on full-information algorithms are quite powerful, and are very different in nature from the more information-theoretic arguments we use in the rest of the thesis (where nodes are restricted to messages of bounded size).

Our main tool in Chapter 4 is *epistemic logic*, the logic of knowledge and belief. By analyzing the state of the nodes' knowledge about an execution we are able to prove a set of powerful lower bounds:

- Consensus, even with no timing constraints, is as hard as hearing from everyone, which means that it is as hard as computing the minimum (or other duplication-insensitive functions).
- *Simultaneous* consensus, which adds the requirement that all nodes halt at the same time, requires n rounds in *every* execution, even the static clique graph. This lower bound is shown by analyzing the time required for nodes to acquire *common knowledge* [48].
- If we compromise and allow nodes to halt within Δ rounds of each other, we make non-trivial gains compared to simultaneous consensus (slashing the running time by a factor of 2, for example), but we still have a lower bound of $n - O(\Delta^{0.28} n^{0.72})$ rounds in line graphs (with *any* input assignment), and a lower bound of $\Omega(nD/(D + \Delta))$ rounds in graphs of dynamic diameter D.

1.3.5 Finding Spanning Trees and Solving Task Allocation

In *undirected* networks, most any "interesting" function of the input can be computed by first constructing a BFS tree of the network, and then computing the value of the function "up the tree". This requires only $O(D)$ rounds (or sometimes more, for "holistic" functions like the median or the mode). Our results in Chapter 6 show that the situation is much more complicated in directed networks: we cannot compute easy functions like minimum or sum in $O(D)$ rounds, and instead they requires poly(n/B) rounds, independent of the diameter.

Motivated by this observation we ask: just what *is* the round complexity of finding a rooted spanning tree in a directed network? As we said, many functions can be computed very quickly by convergecast up a tree (provided the tree is of low depth). If we want compute several consecutive easy functions instead of just one, is it perhaps worth investing in finding a spanning tree first, and then computing our functions by convergecast?

The answer is a fairly resounding "no": we show that finding a rooted spanning tree requires $\Omega(n/B)$ rounds, even in networks of diameter 2, where the size of the network and the UIDs of the nodes are known in advance, even when there are no long simple paths that might "confuse" the spanning tree algorithm. This means that in the time required to compute the spanning tree once, we could instead compute $O(\sqrt{n/B})$ separate minima, using the algorithm from Chapter 6. We obtain a hint of this result in Chapter 6, but to show the general statement, in Chapter 7 we introduce a new communication complexity problem, called TASKALLOC.

The TASKALLOC problem can be viewed as a simple distributed variant of the well-known k-server problem in combinatorial optimization [52]. We have *m players*, and each player initially receives a set of *tasks* drawn from $\{1, \ldots, n\}$. The goal is for the players to partition the tasks amongst themselves, so that each player eventually outputs a set of tasks that is disjoint from the sets output by the other players, and together the players cover all the tasks.

The connection between TASKALLOC and finding a rooted spanning tree is not hard to see: in order to construct the spanning tree, each node must choose a subset of its incoming edges and claim their sources as its children. We do not allow more than one node to claim any given node as a child, and we also require each node (except the root) to be claimed by some parent. Therefore a lower bound on TASKALLOC immediately translates to a lower bound on spanning tree computation. In Chapter 7 we give a lower bound of $\Omega(n)$ on the randomized two-player communication complexity of TASKALLOC, and deduce the $\Omega(n/B)$-round lower bound discussed above on finding a spanning tree.

We believe that the TASKALLOC problem is interesting for its own sake, as it models a job-allocation scenario often encountered in distributed systems. In Chapter 7 we also give several algorithms for solving this problem in various settings, from the classical shared-blackboard model of communication complexity, to the general model of directed networks used in the rest of the thesis.

1.4 Related Work

In this section we briefly survey the main research directions in the field of wireless and dynamic networks, and compare them to the approach taken in this thesis. Our discussion here will be kept at a high level; more specific results and connections will be pointed out in the body of the thesis where relevant.

1.4.1 Models for Dynamic Networks

Networks with a dynamic topology arise in many different circumstances: mobile ad-hoc networks are a natural example, but even static wireless or wired networks may experience topology changes as a result of link and node failures. Different causes for dynamic behavior yield drastically different models of dynamic networks.

Static Networks with Faults

Since fault-tolerance is a mainstay of distributed computing, topology changes resulting from faults have been extensively studied since the very beginnings of the field [10, 110]. Many types of faults have been considered [110]: *crash faults*, where nodes simply crash and stop working; *omission faults*, where nodes may fail to send or receive messages, but otherwise work correctly; and *Byzantine faults*, where nodes may behave arbitrarily and maliciously.

When failures are the only cause of topology changes, it is reasonable to assume that changes will be few and far-between; the models used to represent such networks are typically quite restrictive in the type of changes that they allow. However, they do allow for a dynamic topology, and in that sense they can be viewed as dynamic networks models. In particular, the *omission fault model* (e.g., [112]), like our dynamic graph model, involves nodes that are essentially reliable, and communication links that may fail arbitrarily.

One of the most widely-used models is the *f-bounded fault model* [110], in which no more than f nodes may experience faults that cause them to crash or fail to send or receive messages. It is well-known that in asynchronous networks, even a single faulty node can render simple tasks such as consensus impossible [54], but under various assumptions of synchrony or partial-synchrony meaningful computation is possible with up to $n - 1$ faults (e.g., [104, 44, 134]). Another assumption, studied for example in [88, 90, 111], requires topology changes to be infrequent and spread out over time, so that the system has enough time to recover from a change before the next one occurs. Some of these algorithms use link-reversal [59], an algorithm for maintaining routes in a dynamic topology, as a building block.

Another popular assumption for studying fault-prone networks is *eventual stabilization* (e.g., [3, 15, 16, 141, 75]), which asserts that although the network may be arbitrarily badly-behaved at the beginning of the execution, changes eventually stop occurring. Algorithms developed for this setting typically guarantee safety throughout the execution, but progress is only guaranteed to occur after the network stabilizes. In this context, *self-stabilization* is a useful property that requires that the system converge to a valid configuration from any arbitrary starting state. We refer to [40] for a comprehensive treatment of this topic.

In contrast to all the models above, the dynamic graph model we study in this thesis does not restrict the number, frequency, or duration of changes to the network. Our only assumption is that the network remains connected at any given time; subject to this constraint, arbitrarily many links may appear and disappear, and the network is not assumed to eventually stabilize. However, we do assume that the participants in the computation are reliable: they always follow the protocol, and do not crash. In this sense our model is incomparable with the bounded crash-fault and Byzantine-fault models [110].

Population Protocols

The *population protocol model*, introduced in [5], models a dynamic system as a collection of finite-state agents that interact whenever a pair of agents "meet" and communicate. The model is intended to represent large networks of passive and computationally-weak sensors, such as might be deployed, for example, to track the behavior of birds in the wild [5]. We refer to [8] for a comprehensive survey of work on population protocols.

The population protocol model differs from our dynamic graph model in several fundamental ways:

- Although the schedule of pairwise interactions between agents is controlled by a worst-case adversary, population protocols typically rely on a *strong global fairness* assumption which requires every pair of agents to interact infinitely often in an infinite execution. In contrast, our model assumes that any connected topology may arise, and two nodes may potentially never communicate with each other directly.

- The agents in a population protocol are extremely weak computationally: they are *anonymous*, *identical* and have a *constant amount of memory* that does not grow with the network size. This severely restricts the power of the model: it is known that the original model introduced in [5] is able to compute exactly the set of semilinear predicates [6]. Subsequent work has considered removing some or all of these restrictions (see [8]), obtaining models ranging in strength from semilinear predicates all the way up to linear-space Turing machines.

  In this thesis we make no computation restrictions on the nodes participating in the computation; each node has a unique identifier and may locally store arbitrary amounts of information, although in practice our algorithms tend to require sublinear space.

- Population protocols compute some function *in the limit*: all agents must eventually stabilize on the correct answer, but they do not have to *know* when they have it. This can make sequential composition of protocols challenging, since it is not possible to execute a protocol until it terminates, then take the final result and use it as input for some other computation. Instead, one may use *self-stabilizing* population protocols, which are resilient to inputs that fluctuate and eventually stabilize to some value; but this is not always possible [53].

  In our model nodes must eventually output the result of the computation, and sequential composition is straightforward, as we will see in many instances throughout this thesis.

Dynamic Overlay Networks

Our dynamic graph model assumes the network topology is controlled by an adversary, but there is a vast literature on algorithms that choose the communication topology themselves, typically in the form of an *overlay network* that the algorithm superimposes on an underlying fully-connected network. One very popular application domain is *peer-to-peer protocols*, where participants must maintain a small number of connections to peers they select, e.g., [127, 138]. The algorithm must deal with nodes joining and leaving the network (*churn*) by adjusting the overlay network. Examples of protocols that deal with continual concurrent joins and leaves controlled by an adaptive worst-case adversary are given, for instance, in [97, 107].

A very popular line of research on dynamic overlay networks is *gossip protocols*, which follow the following basic template: the underlying network topology is assumed fully-connected, but in each round, each node chooses a random neighbor, and either sends its information to that neighbor (*push*) or asks that neighbor for its information (*pull*). Basic gossiping protocols can be used to efficiently disseminate information or compute aggregates such as the sum or maximum of inputs (e.g.,[82, 85, 84, 114, 73] and many others). In order to make gossip algorithms more scalable, it is possible to maintain a sparse random overlay structure, and have nodes contact a random neighbor from the overlay graph rather than the entire network [61, 66].

When the underlying network topology is not fully-connected, the performance of gossip algorithms typically depends on the *conductance* of the underlying graph [30, 64]; informally, the conductance measures the sparsity of cuts in the graph. However, it was recently shown that gossip-based techniques can be used to quickly emulate local broadcast in any static graph [28], regardless of its conductance.

Geometric and Random Mobility Models

Much of the work on dynamic networks is motivated by an underlying distribution of the nodes on the two- or three-dimensional plane, with some geometric constraint governing which nodes have a communication link to each other. Perhaps the most popular assumption is the *unit disk graph model*, where only nodes within some fixed distance of each other can directly communicate; see [100] for a discussion of this model and its generalization, the *quasi-unit disk graph model*.

In geometric mobility models, it is assumed that from time to time the nodes change their positions according to some *mobility pattern*, either adversarially controlled or random; e.g., the random way-point model [22, 79, 105], the models surveyed in [27], and the work of [35, 34], where nodes are assumed to travel randomly over a square grid with some fixed speed. In [49, 50, 51], simulation results are used to analyze the performance of flooding-based broadcast and routing protocols in various random geometric models.

Another line of work considers graphs that are not motivated by a geographic model, but evolve randomly over time. Random graph models are typically *Markovian*: the dynamic graph for round $r + 1$ is chosen according to some probability distribution that only depends on $G(r)$ (see, e.g., [11]). A special class of Markovian graphs has been considered in [21, 32, 35]: there it is assumed that in each round, each edge appears or disappears with some fixed probabilities p and q (respectively) independent of other edges.

Algorithms for randomly-changing graphs differ significantly from the algorithms we consider in this thesis, because many random graph models studied in the literature enjoy good expansion, quick propagation of messages, and other nice density and connectivity properties. Sometimes the random graph converges to a known and "easy" stationary distribution; for instance, the Markovian birth-death graph model considered in [76] is shown to converge to the Erdős–Rényi random graph $G_{n,p}$, where each edge exists with probability p independent of other edges. These properties allow one to come up with fast and elegant algorithms, but it is not clear to what extent real mobile networks obey the assumptions of random graph models. For example, is it reasonable to assume that communication links originating at the same node appear or disappear independently of each other? We avoid such questions by allowing the graph topology to change arbitrarily, rather than assuming some "nice" distribution.

Models with Arbitrary Topology Changes

Protocols that work in the presence of continual and arbitrary dynamic changes have not been widely studied prior to our work. Early work (e.g., [14]) considered the problem of end-to-end message delivery in continually changing networks under an assumption of *eventual connectivity*, which asserts that the source and the destination are connected by a path whose links appear infinitely often during the execution. There is also some work on handling nodes that join and leave continually in peer-to-peer overlay networks [71, 97, 106]. Most closely related to the problems studied here is [119], where a few basic results in a similar setting are proved; mainly it is shown that in 1-interval connected dynamic graphs (the definition in [119] 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 [4], Afek and Hendler give lower bounds on the message complexity of global computation in asynchronous networks with arbitrary link failures.

1.4.2 Computation in Wireless Networks

Many of the computation tasks we study in this thesis have previously been studied in the context of static networks. We now review existing literature on the main problems we will discuss.

Consensus, Simultaneous Coordination, And Knowledge

Consensus is a central topic in distributed computing, initiated by the seminal paper by Pease, Shostak and Lamport [124]. Most of the literature on the subject in the context of message-passing systems assumes that the network is a complete graph, with direct channels connecting every pair of nodes. For more general networks, there has been work on the connectivity requirements for reaching consensus under various failure models (see, e.g., [39]), as well as work on implementing consensus in bounded-degree networks with special properties, such as expanders [63, 46]. We are not aware of a study of the efficiency of consensus protocols in general graphs, much less general graphs with a dynamic topology.

While most of the literature on consensus is concerned with tolerating node failures, in the dynamic network model that we consider here the nodes themselves are assumed to be reliable, but the protocol must overcome potentially drastic changes in topology between rounds. Santoro and Widmayer studied consensus in the context of edge failures [132], and showed that it is unsolvable if more than $n - 2$ (arbitrarily chosen) edges can be down in every round. The dynamic network model allows a much broader set of executions, since almost all (in fact, all but $n - 1$) edges can be down in every round, and their choice is almost arbitrary. The only requirement is that the network in each round be connected.

Some of our results on consensus concern cases in which the number of nodes in the network is unknown, or in which there is a rough but inexact bound on the number of nodes. These are unusual assumptions in the context of consensus. A number of standard consensus protocols (e.g., [17]) can easily be modified to handle such assumptions, but this is only due to the fact that the network there is a complete graph, so that a node hears from all correct nodes in every round.

In Chapter 4 we also consider the problem of *simultaneous coordination*. Simultaneous coordination has been shown to be closely related to the notion of common knowledge [69, 48]. Thus, for example, in a simultaneous consensus protocol [45, 113], when the nodes decide on v, it must be common knowledge that some initial value is v. This is much stronger than for regular consensus, in which a node deciding v must (individually) know that one of the initial values was v. It has been shown that deciding in simultaneous consensus (and in a large class of simultaneous coordination tasks) can be reduced to the problem of computing when facts (and which facts) are common knowledge at any given point in an execution. For simultaneous tasks, this enables the design of protocols that are *all-case* optimal: for *every* behavior of the adversary, in the execution of the all-case optimal protocol, nodes decide as fast as they do for that behavior under any other protocol. (All-case optimality does not exist for eventual consensus, as shown in [113].)

Part of our analysis in Chapter 4 centers on the problem of Δ-coordinated consensus, in which decisions must be taken at most Δ rounds apart. In the standard literature, many protocols for eventual agreement are 1-coordinated in this sense: because the network is assumed to be fully-connected, once some correct node v decides, all other correct nodes find out about v's decision in the next round; it is then safe for all correct nodes to decide v as well. For networks that are general graphs, we know of no work developing Δ-coordinated consensus protocols. As in the case of simultaneous coordination, the property of Δ-coordination has a natural counterpart in knowledge theory, called Δ-common knowledge. Very roughly speaking, if u knows that a fact is Δ-common knowledge, then within Δ rounds everyone will know that this is the case. In order to decide, a node must know that the decision value is Δ-common knowledge [69, 48]; the analysis in Section 4.4 is the first case in which such coordination is analyzed and nontrivial bounds are obtained as a result.

Data Aggregation and Spanning Tree Computation

In Chapters 6 and 7 we study the complexity of computing functions of the initial inputs and of finding a rooted spanning tree of a static network. Early work on these problems was concerned with their *message complexity*, that is, the total number of messages sent by all processes, as well as their time complexity. Awerbuch observed in [12] that in undirected networks, the message and time complexity of leader election, computing a distributive sensitive function (e.g., minimum or maximum) and counting are all within a constant factor of the complexity of finding a spanning tree in the network.

It is also shown in, e.g., [12, 56] that the time complexity of these problems in undirected networks is $\Theta(n)$ and the message complexity is $\Theta(m+n\log n)$ in networks of size n with m edges. However, the $\Omega(n)$ lower bound is obtained in networks of diameter $\Omega(n)$, and the message complexity lower bound does not yield a non-trivial bound in our model. In a synchronous undirected network of diameter D edges, it is possible to construct a breadth-first search spanning tree in $O(D)$ rounds, even if the diameter and size of the network are not known in advance. Using such a tree, functions such as minimum, maximum, sum, or average can all be computed in time $O(D)$. Based on a pre-computed spanning tree, many papers have also considered the computation of more complicated functions such as the median or the mode [91, 92, 117, 123, 130, 131, 136].

A related problem that has drawn particular interest in the distributed computing community is computing a *minimum-weight spanning tree* (MST). Since the seminal paper of Gallager, Humblet and Spira [60], the problem has been studied extensively, typically in static and undirected networks (e.g., [12, 62, 47] and others). Of particular relevance to our work here is the lower bound given in [133], which is based on a reduction from the two-player communication problem of Set Disjointness. Communication complexity lower bounds will be used extensively in Chapter 6 of this thesis, but the style of reduction we will use is quite different from reductions used in existing literature (see the discussion in Section 5.2).

Broadcast and Information Dissemination

Another class of widely-studied problems involves broadcasting information throughout the network and collecting information from other nodes. Early work showed that using pipelining, k pieces of information can be disseminated throughout a static network with a latency of at most k [139]; in this thesis we extend pipelining to dynamic networks and use this technique extensively. Another technique often used for information dissemination is *gossip*, which we discussed in Section 1.4.1.

The problem of broadcasting a single message or multiple messages throughout the network has been studied in many different contexts, and in particular in wireless networks with collisions ([18, 89, 36, 93], to list but a few examples of the vast literature on this topic). This work is markedly different from our work here, because it assumes that when two nodes send a message to the same target node at the same time, the messages may *collide* and fail to arrive at their destination; in contrast we assume that nodes can receive arbitrarily many messages in one round. The time required for global broadcast has been studied in a probabilistic version of the edge-dynamic graph model, where edges are independently formed and removed according to simple Markov processes [21, 32, 33]. Similar edge-dynamic graphs have also been considered in control theory literature, e.g. [120, 129]. In [33] the authors also consider a worst-case dynamic graph model which is similar to ours, except that the graph is not always connected and collisions are modelled explicitly. This lower-level model does not admit a deterministic algorithm for global broadcast; however, [33] gives a randomized algorithm that succeeds with high probability.

1.5 Organization of the Results

This thesis is divided into two parts. In Part I we study the limits of computation in highly-dynamic networks. We present our dynamic graph model in Chapter 2, and discuss its basic properties and the rules characterizing information flow in dynamic graphs. In Chapter 3, we study the high-level tasks of *information dissemination* and *counting*; we show that it is possible to solve these tasks in dynamic graphs even under minimal assumptions, and study ways in which performance can be improved using techniques such as pipelining or randomization. The results in Chapter 3 are concerned with algorithms that use only small messages, of polylogarithmic size.

In Chapter 4 we turn to a more abstract investigation of *knowledge* and *common knowledge* in dynamic graphs, with the goal of analyzing the tasks of *consensus* and *simultaneous consensus*. Our main tool in this chapter is *epistemic logic*. We give lower bounds on the ability of algorithms to acquire knowledge, common knowledge, and Δ-approximate common knowledge, and leverage these bounds to obtain lower bounds on the tasks of consensus, simultaneous consensus and Δ-coordinated consensus (respectively). On the positive side, we give a general scheme that allows one to transform a consensus algorithm into a Δ-coordinated consensus algorithm (adding the constraint that all nodes should halt within Δ rounds of each other), and give two concrete examples.

In Part II of the thesis we restrict attention to *static networks*, and study the effects of *asymmetric communication channels*, which can easily arise in wireless networks. In this part of the thesis we rely on lower bounds and techniques from the world of *communication complexity*, which we review in Chapter 5. Next, in Chapter 6, we study the round complexity of computing data aggregates in directed networks of small diameter, and give several surprising lower bounds which show that these tasks are much harder in directed networks than they are in undirected networks. Finally, in Chapter 7, we introduce a new problem, *task allocation*, which models a scenario where the network nodes must partition a set of tasks between them. We give a strong lower bound on the two-player communication complexity of task allocation, and use it to obtain a lower bound on the round complexity of computing rooted spanning trees in shallow networks. We also give several algorithms for solving task allocation in the shared-blackboard model and also in the network model used elsewhere in the thesis.

1.6 Definitions and Notation

Sets and functions. We use $\mathbb{N}$, $\mathbb{N}^+$, $\mathbb{R}$ and $\mathbb{R}^+$ to denote the natural numbers, the positive naturals, the real numbers and the positive reals, respectively. If A and B are sets, the notation $A \subset B$ indicates that A is a strict subset of B (we use $A \subseteq B$ to denote non-strict inclusion). We use A^B to denote the set of all partial functions from B to A.

Given a function $f : \mathbb{N}^+ \to \mathbb{N}^+$, we let $f^{(i)}$ be defined recursively by

$$\begin{aligned} &f^{(0)}(s) := s, \text{ and} \\ &f^{(i+1)}(s) := f(f^{(i)}(s)) + f^{(i)}(s) \text{ for all } i \geq 0. \end{aligned}$$

(Here g^i stands for standard function composition, that is, g^i is g composed with itself i times; in particular, for $i = 0$ we define $g^0(s) = s$.) Equivalently, $f^{(i)}$ can be defined by

$$\begin{aligned} &f^{(0)}(s) := s, \\ &f^{(1)}(s) := f(s) + s, \text{ and} \\ &f^{(i)} := (f^{(1)})^i \text{ for all } i \geq 2. \end{aligned}$$

If X is a set that does not include the special symbol $\perp$, we let $X_\perp$ denote the set $X \cup \{\perp\}$.

Multisets. We use $\mathbb{N}^{\mathcal{D}}$ to denote the set of all multisets over domain $\mathcal{D}$; each multiset $X \in \mathbb{N}^{\mathcal{D}}$ is represented as a partial mapping from elements of $\mathcal{D}$ to their multiplicities. We also sometimes represent multisets as a set of pairs (x, m), where $x \in \mathcal{D}$ and m is the multiplicity of x.

Strictly speaking, a multiset should map each value of $\mathcal{D}$ to its multiplicity, that is, it should be a total function from $\mathcal{D}$ to $\mathbb{N}$. However, to simplify our notation we also admit partial functions, with the understanding that elements outside the domain of the multiset have multiplicity 0.

Given a mapping $f : V \to \mathcal{D}$, we let

$$\text{m-image}(f) := \{(x, m) \mid m = |v \in V : f(v) = x|\} \in \mathbb{N}^{\mathcal{D}}$$

denote the multiset of values occurring in f's image.

Set notation is extended to multisets in the natural way: if $A, B \in \mathbb{N}^{\mathcal{D}}$, we use $A \subseteq B$ to denote the fact that for all $x, y \in \mathcal{D}$ we have $A(x) \leq B(x)$. Strict inclusion, $A \subset B$, is defined similarly: $A \subset B$ iff $A \subseteq B$ and there is some $x \in \mathcal{D}$ such that $A(x) < B(x)$. The *size* $|A|$ of a multiset A is the sum of the multiplicities of all the values in A.

Graph notation. In the sequel, the word "graph" refers to a directed graph $G = (V, E)$, where $E \subseteq V^2$. An *undirected* graph is a graph where for all $(u, v) \in E$ we also have $(v, u) \in E$. The *distance* $\mathrm{dist}_G(u, v)$ from u to v in G is the length of the shortest directed path form u to v.

Given a graph $G = (V, E)$ and a set $S \subseteq V$, we use

$$\Gamma_G(S) := \{v \in V \setminus S \mid \exists u \in S : (u, v) \in E\}$$

to denote the set of nodes outside S that are out-neighbors of some node in S, and we let

$$\Upsilon_G(S) := \{v \in V \setminus S \mid \exists u \in S : (v, u) \in E\}$$

denote the in-neighbors of S. By extension, $\Gamma^d_G(S)$ denotes the set of nodes at out-distance exactly d from some node in S, and similarly for $\Upsilon^d_G(S)$.

For any d,

$$\Gamma^{(d)}_G(S) := \{v \in V \mid \mathrm{dist}_G(u, v) \leq d\} \quad \text{and} \quad \Upsilon^{(d)}_G(S) := \{v \in V \mid \mathrm{dist}_G(v, u) \leq d\}$$

denote the set of nodes at distance at most d from some node in S or distance at most d to some node in S, respectively (including, in both cases, the nodes of S themselves).

In all of our graph notation, we omit the subscript G when it is clear from the context. In part II of the thesis, we often drop the parentheses and write $\Upsilon^d_G(S)$ instead of $\Upsilon^{(d)}_G(S)$.

Part I

Distributed Computation in Dynamic Networks

Chapter 2

The Dynamic Graph Model

In this thesis we study the computation power of dynamic networks. In the interest of obtaining widely-applicable results, we use an abstract model, which does not capture specific details such as the medium-access control (MAC) protocol used in the network, the geographic locations of the network nodes, or the particular trajectories followed by the mobile nodes. Instead, we model the network *above* the MAC layer, and assume that message delivery and node mobility are governed by a *worst-case adversary*.

This chapter introduces the *dynamic graph model*, an adversarial model for dynamic networks, which will be used in the sequel to study the limits of computation in general dynamic networks. In Section 2.1 we present the model and define the *dynamic graph adversary*. Next, Section 2.2 introduces *single-shot tasks*, the general class of computation problems studied in this thesis, and the notion of a *reduction* between tasks. In Section 2.3 we define several properties characterizing the degree of connectivity, information flow and vertex growth in dynamic networks, and show how these properties relate to each other. Finally, in Section 2.4 we review the basic definitions of epistemic logic, and show how it can be applied to reason about information flow in dynamic graphs. (Epistemic logic will be discussed more extensively in Chapter 4; in this chapter we only review the basic definitions.)

2.1 Dynamic Graphs: an Abstract Model of Dynamic Networks

To represent the evolution of a communication network over time we use a *dynamic graph*, defined as follows.

Definition 2.1 (Dynamic graphs). *A dynamic graph is a pair $G = (V, E)$, where V is a fixed set of nodes and $E : \mathbb{N}^+ \to \mathcal{P}(V^2)$ is an* edge function *assigning a set of (directed) edges to each communication round $r \in \mathbb{N}^+$. Each vertex $v \in V$ represents a* wireless node *participating in the computation, and an edge $(u, v) \in E(r)$ represents a communication link between node u and node v in round r.*

We say that a dynamic graph is *undirected* if for all $r \in \mathbb{N}^+$, if $(u, v) \in E(r)$ then $(v, u) \in E(r)$ as well.

Since we do not seek to model the dynamics of a particular network, we assume that the dynamic graph is generated by an *adversary*, which chooses the edges $E(r)$ for each round r. We distinguish between two types of adversaries:

- An *adaptive* adversary generates the dynamic graph on-the-fly; it chooses the edges $E(r)$ for round r based on the entire history up to round r, including the results of coin tosses in rounds $1, \ldots, r-1$. (We note that at the beginning of each round, nodes may toss additional coins and use the results to determine their behavior in that round. The results of coin tosses made at the beginning of round r are not available to the adversary when it chooses $E(r)$.)

- An *oblivious* adversary must commit to the dynamic graph in advance, choosing $E(r)$ for all $r \in \mathbb{N}^+$ before the first round begins.

For deterministic algorithms this distinction is moot: the only difference between an adaptive adversary and an oblivious adversary is that the latter cannot see the results of coin tosses performed during the execution, but a deterministic algorithm does not use random coins.

Computation model. Since we are interested in wireless networks, we assume that nodes communicate with each other using *local broadcast* (henceforth referred to as simply "broadcast"). Computation is synchronous, and we model it as a sequence of *communication rounds*. At the beginning of each round r, each node u tosses private coins and generates a message m to broadcast in that round. At the same time and independently, the adversary chooses the edges $E(r)$ for round r. (If the adversary is oblivious, it must commit to the edges $E(r)$ in advance; however, the nodes still do not know which edges will appear in round r until round r takes place.) The adversary then delivers u's message to all the nodes it selected as u's out-neighbors in round r, i.e., to the nodes $\{v \in V \mid (u, v) \in E(r)\}$. Finally, the nodes process the messages they received, transition to a new state, and the next round begins.

Synchronous vs. asynchronous wakeup. At the beginning of the execution all nodes are asleep; computation is initiated when the adversary wakes up some subset of nodes. If *synchronous wakeup* is assumed, the adversary must wake up all nodes at once. On the other hand, if *asynchronous wakeup* is assumed, the adversary can initially wake up any non-empty set of nodes; in later rounds the adversary can spontaneously wake up sleeping nodes. If node u is asleep at time r, and u receives a message in round $r + 1$ (from some node that was already awake at time r), then node u wakes up at time $r + 1$. Its state at time $r + 1$ can depend on the messages it receives in round $r + 1$. Node u can send its first message in round $r + 2$.

In the sequel we assume synchronous start unless we state otherwise.

Unique identifiers, input size and message size. Each network node has a unique identifier (UID) drawn from some large well-ordered set $\mathcal{U}$; we assume w.l.o.g. that $\mathcal{U} \subseteq \mathbb{N}$, that is, we think of UIDs as natural numbers. We conflate a graph vertex $v \in V$ with the UID of the wireless node represented by the vertex, and use $\mathcal{G}_{\mathbb{N}}$ to denote the set of all dynamic graphs over nodes drawn from $\mathbb{N}$.

In addition, nodes may receive some input pertaining to the task they must solve. Our algorithms are agnostic to the representation of UIDs and input values; UIDs and input values are treated as "black boxes", whose values can be stored, copied and compared, but not manipulated or combined in any way. In particular, the size of the representation is not available to the algorithm, and nodes cannot infer an upper bound on the size of the network by examining their own UID and the number of bits used to represent it. (This allows, for example, the use of variable-width encoding for UIDs and inputs.) In the sequel, whenever we say that an algorithm *does not require an priori bound on the size of the network*, we mean that it treats UIDs and inputs as black-boxes as explained above, and works correctly when deployed in a network of any size.

In our algorithms nodes typically send each other only a constant number of UIDs and inputs per message, so the message size is linear in the size of the UID and the input (unless noted otherwise). To avoid introducing the size of the input and UID as a parameter in our analysis, we assume that UIDs and inputs are drawn from some domain whose size is polynomial in the size n of the network, and represented using $O(\log n)$ bits. This is not an inherent limitation of our algorithms; if the UID and the input are instead drawn from some set $\mathcal{I}$ whose size is not polynomially related to n, we can replace the $O(\log n)$ term with $O(\log |\mathcal{I}|)$.

2.2 Single-Shot Computation Tasks

We now present the class of computation problems we will study throughout this thesis, *single-shot tasks*, and introduce the notion of a *reduction* between tasks.

Tasks are defined over a domain $\mathcal{D}$. An *input* or *output* over $\mathcal{D}$ for a graph $G = (V, E)$ is a mapping $f : V \to \mathcal{D}$; we say that the nodes of G receive input $I : V \to \mathcal{D}$ if each node $u \in V$ receives $I(u)$ as its initial input, and we say that the nodes of G produce output $O : V \to \mathcal{D}$ if each node $u \in V$ eventually halts and produces $O(u)$ as its output.

The tasks we are interested in are usually UID-agnostic: the task specification is not concerned with the specific value received or produced by each node, but rather the *multiset of values* received or produced by all nodes together. To specify a task in a manner that does not depend on UIDs we use *dynamic graph isomorphisms*, which extend the standard notion of a (static) graph isomorphism in the natural way: a mapping $f : V_1 \to V_2$ is an *isomorphism* between $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ if f is an isomorphism between $(V_1, E_1(r))$ and $(V_2, E_2(r))$ for all $r > 0$. We use $G_1 \sim G_2$ to denote the fact that G_1 and G_2 are isomorphic.

In addition to the output values, we sometimes wish to constrain the times at which nodes halt and produce their output. To formalize such requirements we introduce *timing constraints.* Informally, a timing constraint maps dynamic graphs to a set of permissible halting times for the nodes of the graph. (For convenience we represent timing constraints as relations rather than functions.)

Definition 2.2 (Timing constraints). *A* timing constraint *is a relation $\mathcal{T} \subseteq \mathcal{G}_{\mathbb{N}} \times \mathbb{N}^{\mathbb{N}}$, such that*

(a) If $(G, h) \in \mathcal{T}$, where $G = (V, E)$, then $\text{domain}(h) = V$.

(b) $\mathcal{T}$ is invariant under isomorphism*: if f is an isomorphism from G_2 to G_1, and $(G_1, h) \in \mathcal{T}$, then we also have $(G_2, h \circ f) \in \mathcal{T}$.*

We say that an algorithm $\mathcal{A}$ satisfies *timing constraint $\mathcal{T}$ (resp.* satisfies $\mathcal{T}$ with high probability*) if whenever $\mathcal{A}$ is executed in dynamic graph $G = (V, E)$, for all $H \in$* domain($\mathcal{T}$) *such that $G \sim H$ we have $(H, h \circ f) \in \mathcal{T}$ (resp. with high probability), where f is an isomorphism from H to G and $h : V \to \mathbb{N}$ is a mapping representing the halting time of each node in G (i.e., node $u \in V$ halts at time $h(u)$).*

Definition 2.3 (Single-shot tasks). *A* single-shot task *over a domain $\mathcal{D}$ is a pair $(R, \mathcal{T})$, where $R \subseteq \mathbb{N}^{\mathcal{D}} \times \mathbb{N}^{\mathcal{D}}$ is a relation on multisets, called* the input/output relation *of the task, and $\mathcal{T}$ is a timing constraint. We say that a distributed algorithm $\mathcal{A}$* solves task $(R, \mathcal{T})$ *if*

(a) For all dynamic graphs $G = (V, E)$ and all inputs $I : V \to \mathcal{D}$ in the domain of R, when $\mathcal{A}$ is executed in G with input I, all nodes eventually halt and produce an output O such that (m-image(I), m-image(O)) $\in R$. *Moreover,*

(b) $\mathcal{A}$ satisfies $\mathcal{T}$.

Definition 2.3 above is for deterministic algorithms; it is extended to randomized algorithms in the usual manner, by requiring that $\mathcal{A}$ satisfy $\mathcal{T}$ and produce a correct output on R with high probability (over its own random choices). We require deterministic termination, that is, all nodes must halt in every execution regardless of the outcomes of random choices.[1]

[1]This restriction is without loss of generality: we will see in Chapter 3 that it is possible to deterministically compute the size of 1-interval connected network from no prior information. It follows that if we are given a randomized algorithm $\mathcal{A}$ where with probability $p(n)$ all nodes halt after $t(n)$ rounds and produce a correct output, we can convert it into an algorithm $\mathcal{A}'$ where nodes always halt after at most $2t(n)$ rounds, and a correct output is produced with probability $p(n)$. We do this by interleaving the steps of $\mathcal{A}$ with the counting algorithm from Chapter 3: in every even step we execute one step of $\mathcal{A}$, and in every odd step we execute one step of the counting algorithm. Eventually, at every node, one of the two algorithms halts (because the counting algorithm always halts). If $\mathcal{A}$ finished first, we simply output the result of $\mathcal{A}$. On the other hand, if the counting algorithm finished first, we now know n; we continue running $\mathcal{A}$ until it has taken its $t(n)$-th step (i.e., until time $2t(n)$). If $\mathcal{A}$ does not halt by time $2t(n)$ the node halts and outputs an arbitrary value. From the probabilistic correctness guarantee of $\mathcal{A}$ we know that with probability $p(n)$, $\mathcal{A}$ does finish by time $2t(n)$ at all nodes and produces a correct global output.

Many of the tasks studied in this thesis do not have any non-trivial timing constraints; we say that a task is *untimed* if its timing constraint is

$$\{(G = (V, E), f) \mid \text{domain}(f) = V\},$$

which allows nodes to halt at any time. Informally we also say that untimed tasks *have no timing constraints*, and we conflate untimed tasks with their input/output relations.

A non-trivial timing constraint is often used to impose some degree of coordination upon the nodes. For example, in Chapter 4 we will study *simultaneous tasks*, where the timing constraint is

$$\mathcal{T}_{\mathsf{simult}} := \{(G = (V, E), f) \mid \text{domain}(f) = V \text{ and for all } u, v \in V,\ f(u) = f(v)\}.$$

This constraint requires all nodes to halt at the same time. Another task with a non-trivial timing constraint, HF_m, is introduced in Section 2.3 below; informally, the HF_m task requires nodes to halt only after they have "heard from" at least m nodes, where "u hears from v" if there is a chain of messages leading from v to u (see Definition 2.9).

Task classes. We classify tasks according to the following characteristics of their input/output relation R.

- **Single-valued tasks**: a task where all nodes are required to output the same value. Formally, for each $(X, Y) \in R$, the multiset Y contains a single value whose multiplicity is $|X|$.

- **Deterministic tasks** (or *functions*): a task with input/output relation R is *deterministic* if R is a function from $\mathbb{N}^{\mathcal{D}}$ to $\mathbb{N}^{\mathcal{D}}$. In this case we let $R(X)$ denote the value of R on multiset X. We are often interested in functions R that are single-valued.

- **Globally-sensitive functions**: a function R is said to be *globally sensitive* if we can change the input to a single node and obtain a different output value. Formally, R is globally sensitive if for each n, there exists a multiset X of size n in the domain of R, such that for any multiset X' of size n that differs from X on the input to a single node, we have $X' \in \text{domain}(f)$ and $R(X) \neq R(X')$.

- **ε-sensitive functions**: a generalization of globally-sensitive functions; a function R is said to be *ε-sensitive* if we can change the input to an ε-fraction of nodes and obtain a different output value. Formally, R is ε-sensitive if for each n, there exists a multiset X of size n, such that for any multiset X' of size n that differs from X by at least $\varepsilon \cdot n$ elements we have $X' \in \text{domain}(f)$ and $R(X) \neq R(X')$.

- **Duplication-insensitive**: a task is said to be *duplication-insensitive* if its input/output relation is not sensitive to the multiplicities of values in the input and output. Formally, a task with input/output relation R is duplication-insensitive if for any four multisets X, X', Y, Y', if $\text{domain}(X) = \text{domain}(X')$, $\text{domain}(Y) = \text{domain}(Y')$ and $(X, Y) \in R$, then we also have $(X', Y') \in R$. If a task is not duplication-insensitive then we say that it is *duplication-sensitive.*

Example 2.4. In this thesis we will study the complexity of the following untimed tasks, defined over the domain of natural numbers ($\mathcal{D} = \mathbb{N}$):

- *Counting*, that is, determining the number of nodes in the network. Phrased as a task, each node receives 1 as its input, and must output the sum over all inputs. This task is deterministic and duplication-sensitive, but it is not globally-sensitive, since the input cannot be changed without altering the size n of the network (i.e., there do not exist X, X' in the domain of R that differ from each other but have the same size).

- *Minimum*: each node receives an input, and must output the minimum of all inputs (if inputs are binary, this is equivalent to taking their Boolean AND). This task is deterministic and globally-sensitive, but it is duplication-insensitive: the output depends only on the values present in the input assignment, not on their multiplicities.

- *Consensus*: each node receives a binary input, and all nodes must output the same binary value, which must be the input to one of the nodes. This task is not deterministic: if the input assignment contains both 0 and 1, both values are permissible as output. Consensus is duplication-insensitive, but since it is not a function, it is not globally-sensitive.

Finally, let us define what it means for one task to be *at least as hard* as another. We are interested in the size of messages and number of rounds required to solve tasks, and we therefore introduce a notion of reduction that preserves these parameters.

Definition 2.5 ((α, β)-reduction). *Let $\alpha, \beta : \mathcal{G}_{\mathbb{N}} \to \mathbb{N}$. An (α, β)-reduction from task T_1 to task T_2 is an algorithm $\mathcal{A}_1$ that solves T_1 using black-box access to an algorithm $\mathcal{A}_2$ for solving T_2. When executed in dynamic graph G, $\mathcal{A}_1$ can interact with $\mathcal{A}_2$ as follows:*

- *$\mathcal{A}_1$ initially specifies an input for $\mathcal{A}_2$, and then runs alongside $\mathcal{A}_2$ until $\mathcal{A}_2$ halts.*

- *During $\mathcal{A}_2$'s run, $\mathcal{A}_1$ cannot examine $\mathcal{A}_2$'s internal state or the contents of its messages. However, $\mathcal{A}_1$ can append at most $\beta(G)$ bits to each message sent by $\mathcal{A}_2$ (treating each message as a black box).*

- *When $\mathcal{A}_2$ halts, $\mathcal{A}_1$ can observe $\mathcal{A}_2$'s output value. $\mathcal{A}_1$ may continue running after $\mathcal{A}_2$ terminates, using messages of size at most $\beta(G)$.*

In executions where $\mathcal{A}_2$ solves T_2 correctly, $\mathcal{A}_1$ must with high probability terminate at most $\alpha(G) + 1$ rounds after $\mathcal{A}_2$ does and solve T_1. (If $\mathcal{A}_1$ is a deterministic reduction, *the above should hold deterministically in every execution where $\mathcal{A}_2$ succeeds.)*

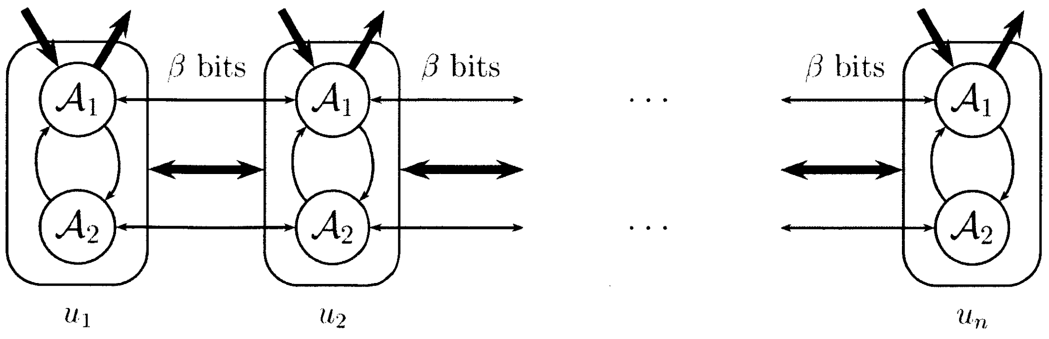


Figure 2-1: A reduction $\mathcal{A}_1$ from T_1 to T_2, combined with an algorithm $\mathcal{A}_2$ for solving T_2. Thick lines indicate the input and output to the system, as well as messages exchanged between nodes. Thin lines indicate the (virtual) interaction between the (virtual) instances of $\mathcal{A}_1$ and $\mathcal{A}_2$ running in the system. Since $\mathcal{A}_1$ cannot interfere with or examine $\mathcal{A}_2$'s execution, we can think of $\mathcal{A}_1$ as executing alongside $\mathcal{A}_2$ and sending its own messages, which are separate from $\mathcal{A}_2$'s messages and comprise at most β bits each. $\mathcal{A}_1$ provides the input to $\mathcal{A}_2$ and captures $\mathcal{A}_2$'s output.

Definition 2.6 (Complete tasks). *We say that a task T is* (α,β)-complete *for a task class $\mathcal{C}$ if any task $T' \in \mathcal{C}$ (α,β)-reduces to T.*

The parameters α, β typically depend only on some parameters of the dynamic graph (e.g., its size or its interval connectivity); to simplify our notation, in such cases we represent α, β as mappings from the graph parameter's value (or from a tuple of parameter values) to natural numbers. Most often α and β depend only on the size n of the graph and we represent them as mappings $\mathbb{N} \to \mathbb{N}$.

We say that T_1 *(α,β)-reduces* (or *(α,β)-reduces via a high-probability reduction*) if there exists a deterministic (α,β)-reduction (resp. a randomized (α,β)-reduction) from T_1 to T_2. If T_1 (α,β)-reduces to T_2, then we can solve T_1 by running an algorithm for solving T_2 in parallel with an (α,β)-reduction from T_1 to T_2 (see Fig. 2-1 for an illustration), obtaining the following relationship between the complexity of T_1 and of T_2:

Proposition 2.7. *If T_2 can be solved (or solved w.h.p.) in $t(n)$ rounds using $B(n)$-bit messages, and T_1 is a task that $(\alpha(n),\beta(n))$-reduces (resp. w.h.p.) to T_2, then T_1 can be solved (resp. solved w.h.p.) in $t(n)+\alpha(n)+1$ rounds using $(B(n)+\beta(n))$-bit messages.*

The reductions in this thesis will mostly fall into the following three classes:

- A subclass of $(\alpha, O(\log n))$-reductions, in which the reduction calls the algorithm for T_2, waits until it terminates without doing any work itself in the meantime, and only then "starts running" for α rounds. We can think of such a reduction as an algorithm that takes as part of its own input a solution to T_2 on input it selects. Typically such reductions will be to *simultaneous* tasks, so that we are assured that all nodes finish solving T_2 and "start running" the reduction at the same time.

- $(0, \beta)$-reductions: a $(0, \beta)$-reduction from T_1 to T_2 is an algorithm $\mathcal{A}_1$ that takes an algorithm $\mathcal{A}_2$ for T_2, appends some extra information to $\mathcal{A}_2$'s messages, and halts immediately when $\mathcal{A}_2$ does. The typical $(0, \beta)$-reduction in this thesis uses $\mathcal{A}_2$ as a *termination condition*: $\mathcal{A}_1$ will not use $\mathcal{A}_2$'s output at all, only the fact that $\mathcal{A}_2$ has terminated.

  To argue the correctness of such a reduction we show that solving T_2 requires $\mathcal{A}_2$ to detect when the dynamic graph satisfies certain conditions, and thus $\mathcal{A}_1$ can "outsource" detecting these conditions to $\mathcal{A}_2$. For example, to compute the minimum input value, we can design an algorithm $\mathcal{A}_1$ where nodes always forward the smallest input value they have heard so far. Eventually the global minimum will be received by all nodes, and at this point $\mathcal{A}_1$ can halt; however, $\mathcal{A}_1$ by itself cannot detect when it should halt. If we have another algorithm $\mathcal{A}_2$ that "knows when to halt" (for example, we will see in Section 3.1 that an algorithm for counting can serve in this role), then $\mathcal{A}_1$ can piggyback on top of $\mathcal{A}_2$'s messages, and halt when $\mathcal{A}_2$ halts.

- $(0, 0)$-reductions: a $(0, 0)$-reduction simply maps inputs for T_1 to inputs for T_2, and then maps the output for T_2 back to an output for T_1. Informally, a $(0, 0)$-reduction asserts that any algorithm for T_2 must implicitly solve T_1.

2.3 Connectivity, Expansion and Information Flow in Dynamic Networks

In static networks, the diameter of the network graph and its connectivity and expansion properties greatly impact the performance of distributed algorithms in the network. We now define several dynamic graph properties that extend these notions to the dynamic case.

Connectivity. Global distributed computation usually requires some notion of *connectivity*: essentially, we must be guaranteed that given sufficient time and effort, a message from any node in $u \in V$ can reach any other node in $v \in V$. In the case of dynamic networks, a basic requirement is *instantaneous connectivity*, that is, each instantaneous graph $(V, E(r))$ for any $r \in \mathbb{N}^+$ must be connected. However, if the network enjoys some stability, connected subgraphs may persist for more than a single round; the following definition captures the *amount* of stability in the network. Recall that dynamic graphs are defined as directed graphs in Definition 2.1.

Definition 2.8 (T-interval connectivity). *A dynamic graph $G = (V, E)$ is said to be* T-interval connected, *for $T \in \mathbb{N}^+$, if for each $r \in \mathbb{N}^+$, the edges $\bigcap_{r' \in [r, r+T-1)} E(r')$ induce a strongly-connected graph on V.*

We abuse this definition slightly by saying that a graph is ∞-*interval connected* if $\bigcap_{r \in \mathbb{N}^+} E(r)$ induces a strongly-connected graph on V.

Our algorithms assume at least 1-interval connectivity (that is, each instantaneous graph must be strongly-connected).

Information flow. To capture the manner in which information propagates in a dynamic network we use a version of *Lamport's causal order* [103], defined as follows.[2]

Definition 2.9 (Lamport's causal order). *Given a dynamic graph $G = (V, E)$, Lamport's causal order for G is a relation $\rightsquigarrow_G \subseteq (V \times \mathbb{N}^+)^2$ indicating the existence of a (directed) chain of messages between two nodes at two points in time. Formally, the relation $\rightsquigarrow_G$ is defined as the transitive and reflexive closure of the relation $\rightarrow_G$, where $(u, t) \rightarrow_G (v, t+1)$ iff either $u = v$ or $(u, v) \in E(t+1)$.*

We omit the subscript G from our notation when the graph is clear from the context.

The causal order has the following simple properties:

Proposition 2.10. *The causal order $\rightsquigarrow$ is reflexive and transitive. Moreover,*

(a) For all $u \in V$ and $t \leq t'$ we have $(u, t) \rightsquigarrow (u, t')$;

(b) For all $u, v \in V$ and $t_1 \leq t_2 \leq t_3$, if $(u, t_1) \rightsquigarrow (v, t_2)$ then $(u, t_1) \rightsquigarrow (v, t_3)$, and if $(u, t_2) \rightsquigarrow (v, t_3)$ then $(u, t_1) \rightsquigarrow (v, t_3)$.

(c) For all $u, v \in V$ and $t \in \mathbb{N}$, if $u \neq v$ then $(u, t) \not\rightsquigarrow (v, t)$.

Proof. Reflexivity and transitivity follow from the definition of $\rightsquigarrow$ as the reflexive and transitive closure of $\rightarrow$. Part (a) follows from transitivity and the fact that $(u, t) \rightarrow (u, t+1)$ (by definition of $\rightarrow$). Part (b) follows from transitivity and part (a). For part (c), suppose that $(u, t) \rightsquigarrow (v, t)$. Then there is a sequence $(u_0, t_0), \ldots, (u_k, t_k)$ such that $(u, t) = (u_0, t_0) \rightarrow (u_1, t_1) \rightarrow \ldots \rightarrow (u_k, t_k) = (v, t)$. By definition of $\rightarrow$, for all $x, y \in V$ we have $(x, t) \not\rightarrow (y, t)$, and moreover, if $t' > t$ then $(x, t') \not\rightarrow (y, t)$. Thus, we can show by induction on k that $u_i = u$ and $t_i = t$ for all $i \leq k$. It follows that $u = v$, as required. □

[2]The causal order was originally defined in [103] for asynchronous systems; the definition of $(u, t) \rightarrow (v, t')$ in [103] asserts that at time t, node u sends a message which v receives at time t'. In contrast, Definition 2.9 above is stated in terms of the dynamic graph G alone, but the two definitions are equivalent: since the network is synchronous, and nodes communicate by local broadcast, node u sends a message to v in round r iff $(u, v) \in E(r)$.

The pairs (u,t) (where $u \in V$ and $t \in \mathbb{N}$) are referred to as *time-nodes.* If $(u,t) \rightsquigarrow (v,t')$, then we say that (u,t) *causally influences* (v,t').

Lamport's causal order yields a pessimistic characterization (from the perspective of proving lower bounds) of the information that nodes can acquire about the execution: if $(u,t) \rightsquigarrow (v,t')$, then at time t' node v may or may not have some information regarding the state of node u a time t, depending on the contents of messages sent by the algorithm; but if $(u,t) \not\rightsquigarrow (v,t')$, then node v *cannot* know any non-trivial fact regarding the state of u at time t, regardless of the messages sent by the algorithm. (We defer the definition of a "non-trivial fact" until Section 2.4.) For example, if $(v,0) \not\rightsquigarrow (u,t)$, then at time t node u cannot distinguish the current execution from one in which node v receives a different input or is not present in the network at all. This fact will be used extensively when we characterize the hardness of computing functions of the nodes' initial input (e.g., the minimum input or the sum of the inputs).

To represent the set of time-nodes influencing a given time-node we introduce the following notation.

Definition 2.11. *The* past set *and* future set *of time-node* (u,t) *are defined by*

$$\mathsf{past}(u,t) := \{(v,t') \mid (v,t') \rightsquigarrow (u,t)\}\,, \textit{ and}$$
$$\mathsf{future}(u,t) := \{(v,t') \mid (u,t) \rightsquigarrow (v,t')\}\,.$$

The projection *of a set of time-nodes $X \subseteq V \times \mathbb{N}$ on a time $t \in \mathbb{N}$ is defined by*

$$X_t := \{v \in V \mid (v,t) \in X\}\,.$$

We can now re-state Proposition 2.10 in terms of past and future sets:

Proposition 2.12. *For all nodes $u, v \in V$ and times $t \le t' \le t''$,*

(a) $\mathsf{past}(u,t) \subseteq \mathsf{past}(u,t')$ *and* $\mathsf{future}(u,t') \subseteq \mathsf{future}(u,t)$.

(b) $\mathsf{past}(u,t'')_{t'} \subseteq \mathsf{past}(u,t'')_t$ *and* $\mathsf{future}(u,t)_{t'} \subseteq \mathsf{future}(u,t)_{t''}$.

(c) $\mathsf{past}(u,t)_t = \{u\}$ *and* $\mathsf{future}(u,t)_t = \{u\}$.

Proof. Parts (a) and (b) follows from part (b) of Proposition 2.10; part (c) combines parts (a) and (c) of Proposition 2.10. □

When we discuss more than one execution or dynamic graph, we distinguish between past sets and future sets in the different executions or graphs by indicating the execution or graph in the subscript; i.e., $\mathsf{past}(u,t)_{\alpha,t'}$ is the projection of the past set of (u,t) in execution α on time t', and similarly for future sets.

The dynamic diameter. In static networks, the efficiency of distributed algorithms is usually closely related to the diameter of the network: the diameter is a bound on the time required for a message from any node to reach all the nodes in the network. In a dynamic network, the diameter of each instantaneous communication graph $(V, E(r))$ does not yield a meaningful bound on the time required for information to propagate throughout the network; instead we introduce the following definition, based on Lamport's causal order.

Definition 2.13 (Dynamic Diameter). *We say that a dynamic graph $G = (V, E)$ has a* dynamic diameter bounded by D *(where $D \in \mathbb{N}^+$) if for any time t and nodes $u, v \in V$ we have $(u, t) \rightsquigarrow (v, t + D)$. The* dynamic diameter *of G is the smallest number $D \in \mathbb{N}$ such that the diameter of G is bounded by D.*

Equivalently, we can define the dynamic diameter of G in terms of past and future sets:

Proposition 2.14. *The dynamic diameter of G is bounded by D iff either of the following conditions holds:*

(a) For every $u \in V$ and $t \geq 0$ we have $\mathsf{future}(u, t)_{t+D} = V$.

(b) For every $u \in V$ and $t \geq D$ we have $\mathsf{past}(u, t)_{t-D} = V$.

Proof. By Definition 2.11 we have $(u, t) \rightsquigarrow (v, t + D)$ iff $u \in \mathsf{past}(v, t + D)_t$ iff $v \in \mathsf{future}(u, t)_{t+D}$. The claim follows. □

Example 2.15. Let us briefly illustrate why this new notion is necessary, and why we cannot simply use a bound on the (standard) diameter of the instantaneous communication graph in each round. Consider the dynamic graph shown in Fig. 2-2. The graph is a "dynamic star" over n nodes $0, \ldots, n-1$: in round r, the center of the graph is node $r \bmod n$, and the remaining nodes are leaves of the star. Suppose that node 0 wishes to disseminate a message m to all the nodes in the network, and that all nodes that receive m cooperate by re-sending the message m in each subsequent round. How long until all nodes receive m?

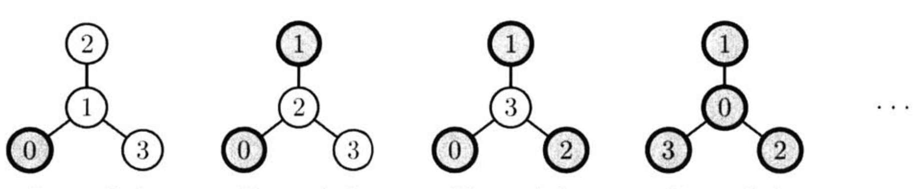


Figure 2-2: The dynamic star graph with $n = 4$. Nodes that received the message are indicated in gray.

Node 0 is initially a leaf, so in the first round its message is received only by the center of the star, node 1. However, in the next round node 1 also becomes a leaf. In the second round, even though nodes 0 and 1 both send m, only node 2, the center of the new star, receives it. In the following round node 2 also becomes a leaf, and so on. In general, at time $i \leq n-2$ the message is known only to leaves of the star, and only one new node (the center of the star) learns it in round $i+1$. It takes $n-1$ rounds for all nodes to receive the message. In other words, the *dynamic diameter* of the dynamic star graph is $n-1$, even though the diameter of each "instantaneous star" is 2.[3]

Vertex growth. Real wireless networks are often dense and well-connected. To quantify the degree of connectivity we define the *vertex growth* of a graph, a natural generalization of the notion of a vertex expander.

Definition 2.16 (Vertex growth). *We say that a static graph $G = (V, E)$ has* vertex growth $\mathrm{g} : \mathbb{N}^+ \to \mathbb{N}^+$ *if for every $S \subseteq V$, if $|S| \leq |V|/2$ then $|\Gamma(S)| \geq \mathrm{g}(|S|)$.*

Example 2.17. If G is an undirected vertex expander with expansion constant α, then G has vertex growth $\mathrm{g}(s) := \alpha s$. If G is k-connected (for $k \leq n/2$), then G has vertex growth $\mathrm{g}(s) := k$.

It is easy to extend the bound on the size of the immediate out-neighborhood of a set into a bound on the size of the d-neighborhood:

Lemma 2.18. *If G has vertex growth g, then for any distance d and set $S \subseteq V$ we have*

$$|\Gamma^{(d)}(S)| \geq \min\left\{\mathrm{g}^{(d)}(|S|), n/2+1\right\}.$$

Proof. The lemma follows from the definition of vertex growth and an easy induction on d: until the size of $\Gamma^{(j)}(S)$ exceeds $n/2$, we can continue to apply the definition of vertex growth (Definition 2.16) to bound the size of $\Gamma^{(j+1)}(S)$ from below. □

Now we can generalize Definition 2.8 to quantify both the connectivity and the duration of spanning subgraphs in a dynamic graph.

Definition 2.19 ((T, g)-interval connectivity). *We say that a dynamic graph $G = (V, E)$ is* (T, g)-interval connected *if for each $r \in \mathbb{N}^+$, the subgraph induced by $\bigcap_{r' \in [r, r+T-1]} E(r')$ is strongly-connected and has vertex growth g.*

Even though vertex growth is defined over static graphs, we often say that a dynamic graph has vertex growth g if it is $(1, \mathrm{g})$-interval connected.

[3]The dynamic star graph is something of a pathological case: the same graph was also used in [11] to show that a random walk can have exponential cover time in a dynamic graph.

The relationship between vertex growth and dynamic diameter. In Example 2.15 we saw that a small diameter for each instantaneous communication graph does not necessarily translate to a small dynamic diameter. However, good vertex growth for each instantaneous graph does: it implies that in each round, many new nodes are causally influenced by each time-node, and so it does not take many rounds for a pair of nodes to causally influence each other. More formally we can state the following.

Lemma 2.20. *Let $G = (V, E)$ be an undirected dynamic graph over n nodes with vertex growth g. Then the dynamic diameter of G is at most $2d$, where $d = d(\mathrm{g}, n)$ is the smallest integer such that $\mathrm{g}^{(d)}(1) > n/2$.*

Proof. Fix two nodes $u, v \in V$ and a time $t \geq 0$. We must show that $(u, t) \rightsquigarrow (v, t + 2d)$. For each time i, $0 \leq i \leq 2d$, let

$$F_i := \mathsf{future}(u, t)_{t+i} = \{w \in V \mid (u, t) \rightsquigarrow (w, t + i)\},$$

and let

$$P_i := \mathsf{past}(v, t + 2d)_{t+2d-i} = \{w \in V \mid (w, t + 2d - i) \rightsquigarrow (v, t + 2d)\}.$$

(See Fig. 2-3 for an illustration.) We will show that $F_d \cap P_d \neq \emptyset$, that is, there is some node w such that $(u, t) \rightsquigarrow (w, t + d)$ and $(w, t + d) \rightsquigarrow (v, t + 2d)$. This will then imply that $(u, t) \rightsquigarrow (v, t + 2d)$, by transitivity of the causal order.

To show that $F_d \cap P_d \neq \emptyset$, we show by induction on $i \leq d$ that

(I) Either $|F_i| > n/2$ or $|F_i| \geq \mathrm{g}^{(i)}(1)$, and similarly,

(II) Either $|P_i| > n/2$ or $|P_i| \geq \mathrm{g}^{(i)}(1)$.

In other words, as we go forward or back in time from times t and $t + 2d$ (resp.), $F_{t'}$ and $P_{t'}$ expand at the rate of the vertex growth g until their size exceeds $n/2$. In particular, since $\mathrm{g}^{(d)}(1) > n/2$ by definition, we obtain $|F_{t+d}| > n/2$ and $|P_{t+d}| > n/2$, and hence $F_{t+d} \cap P_{t+d} \neq \emptyset$.

The induction base, $i = 0$, is immediate: from Proposition 2.12 we have $|F_0| = |\mathsf{future}(u, t)_t| = 1$ and $|P_0| = |\mathsf{past}(v, t + 2d)_{t+2d}| = 1$, and by definition, $\mathrm{g}^{(0)}(1) = 1$. For the induction step, assume that the claim holds for $i < d$. By Proposition 2.12 we have $F_i \subseteq F_{i+1}$ and $P_i \subseteq P_{i+1}$. If $|F_i| > n/2$, then we have $|F_{i+1}| > n/2$ as well, and similarly for P_i. Thus, assume that $|F_i| \leq n/2$ and $|P_i| \leq n/2$.

Consider the communication graph $G_i := (V, E(t + i + 1))$ in round $t + i + 1$. From the definition of vertex growth, $|\Gamma_{G_i}(F_i)| \geq \mathrm{g}(|F_i|)$. For each node $w \in \Gamma_{G_i}(F_i)$ we have $w \notin F_i$, and there exists some $z \in F_i$ such that $(z, w) \in E(t + i + 1)$. Therefore $(u, t) \rightsquigarrow (z, t + i) \rightsquigarrow (w, t + i + 1)$, and hence $w \in F_{i+1} \setminus F_i$. In addition, $F_i \subseteq F_{i+1}$; therefore $|F_{i+1}| \geq \mathrm{g}(|F_i|) + |F_i| \geq \mathrm{g}(\mathrm{g}^{(i)}(1)) + \mathrm{g}^{(i)}(1) = \mathrm{g}^{(i+1)}(1)$. (In the last step we applied the induction hypothesis, which shows that $|F_i| \geq \mathrm{g}^{(i)}(1)$.)

A similar argument can be made for P_{i+1}, except that we must go "back in time" instead of forward. Let $G_i := (V, E(t + 2d - i))$. Since $|P_i| \leq n/2$, the vertex growth

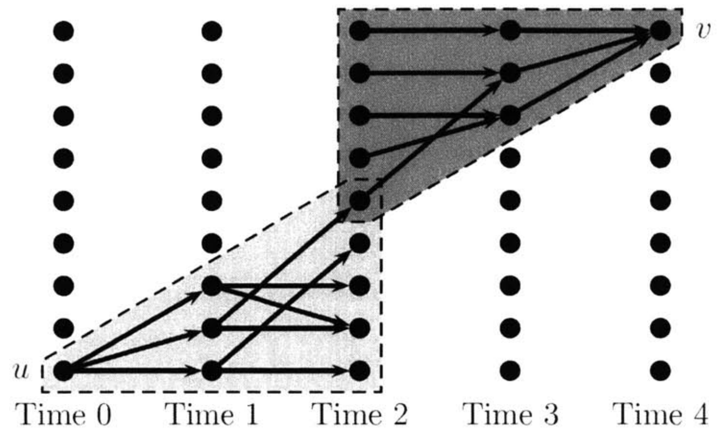


Figure 2-3: The sets $F_{t'}$ and $P_{t'}$ from Lemma 2.20, with $n = 9$ and $t = 0$. Each instantaneous graph is 2-connected (vertex growth 2), but only edges relevant to the proof are shown. The smallest d satisfying $\mathrm{g}^{(d)} > 9/2$ is $d = 2$, and the dynamic diameter is 4. The shaded areas indicate $F_{t'}$ for $t' = 0, 1, 2$ in light gray, and $B_{t'}$ for $t' = 2, 3, 4$ in darker gray, with $F_2 \cap B_2 \neq \emptyset$, as argued in the proof.

of G_i again yields $|\Gamma_{G_i}(P_i)| \geq \mathrm{g}(|P_i|)$. For each node $w \in \Gamma_{G_i}(P_i)$ we have $w \notin P_i$, and there exists some $z \in P_i$ such that $(z, w) \in E(t+2d-i)$. Since G is undirected we also have $(w, z) \in E(t+2d-i)$, and hence $(w, t+2d-(i+1)) \rightsquigarrow (z, t+2d-i) \rightsquigarrow (v, t+2d)$. As above, we obtain $|P_{i+1}| \geq \mathrm{g}(|P_i|) + |P_i| \geq \mathrm{g}^{(i+1)}(1)$. □

We remark that Lemma 2.20 also holds for directed graphs, if their vertex growth applies "in both directions": every set $S \subseteq V$ of size at most $n/2$ has at least $\mathrm{g}(|S|)$ in-neighbors as well as at least $\mathrm{g}(|S|)$ out-neighbors.

Similar arguments to Lemma 2.20 have also been applied to analyze gossip-based algorithms, e.g., in [82, 24]. In a gossip-based algorithm, each node selects a neighbor uniformly at random, and connects to that neighbor. (For example, it might send its information to that neighbor, or request the neighbor's information.) In each round, the graph induced by these connections is with high probability a vertex expander; thus, the *dynamic graph* induced by the algorithm over many rounds has a logarithmic dynamic diameter with high probability, and this is a key factor in the performance of gossip-based algorithms. Lemma 2.20 generalizes the argument slightly in that it allows for other types of expansion (e.g., k-connectivity), but the analysis is quite similar.

One special case of particular interest to us is a 1-interval connected dynamic graph (possibly directed) with no non-trivial vertex growth. Such graphs have a dynamic diameter of at most $n-1$, because the vertex growth $\mathrm{g}(s) := s+1$, which follows from strong connectivity, continues to apply even after the set size s exceeds $n/2$. The following lemma formalizes this intuition and will be quite useful to us in the sequel.

Lemma 2.21. *If G is a 1-interval connected graph over n nodes, then for all nodes*

$u \in V$ and times $t \le t'$ we have

(a) $|\mathsf{past}(u, t')_t| \ge \min\{n, t' - t + 1\}$*, and*

(b) $|\mathsf{future}(u, t)_{t'}| \ge \min\{n, t' - t + 1\}$.

Proof. By induction on $t' - t$. The base case, $t' = t$, is immediate. For the step, assume that the claim holds for all $u \in V$ and all times t', t such that $t' - t = k$, and consider a node $u \in V$ and two timepoints t', t such that $t' - t = k + 1$.

For part (a), observe that $\mathsf{past}(u, t')_{t+1} \subseteq \mathsf{past}(u, t')_t$: if $(v, t + 1) \in \mathsf{past}(u, t')$ then we have $(v, t) \to (v, t + 1) \rightsquigarrow (u, t')$, so $(v, t) \in \mathsf{past}(u, t')$ as well. Thus, if $|\mathsf{past}(u, t')_{t+1}| \ge n$, the claim holds. On the other hand, if $|\mathsf{past}(u, t')_{t+1}| < n$, then from 1-interval connectivity, in the graph $(V, E(t+1))$ for round $t+1$ there is some edge (v, w) in the directed cut $(V \setminus \mathsf{past}(u, t')_{t+1}, \mathsf{past}(u, t')_{t+1})$. We have $v \notin \mathsf{past}(u, t')_{t+1}$ and $(v, t) \to (w, t+1) \rightsquigarrow (u, t')$, so $v \in \mathsf{past}(u, t')_t \setminus \mathsf{past}(u, t')_{t+1}$. Since $t' - (t+1) = k$, the induction hypothesis shows that $|\mathsf{past}(u, t')_{t+1}| \ge t' - (t + 1) + 1 = t' - t$, and therefore $|\mathsf{past}(u, t')_t| \ge t' - t + 1$, as desired.

Part (b) is quite similar. We have $\mathsf{future}(u, t)_{t'-1} \subseteq \mathsf{future}(u, t)_{t'}$, and hence if $|\mathsf{future}(u, t)_{t'-1}| \ge n$ we are done. Otherwise there is some edge (w, v) in the directed cut $(\mathsf{future}(u, t)_{t'-1}, V \setminus \mathsf{future}(u, t)_{t'-1})$ for round t', which implies that $(u, t) \rightsquigarrow (w, t' - 1) \to (v, t')$. Therefore $v \in \mathsf{future}(u, t)_{t'} \setminus \mathsf{future}(u, t)_{t'-1}$, and applying the induction hypothesis yields the claim. □

Corollary 2.22. *The dynamic diameter of any 1-interval connected graph over n nodes is at most $n - 1$.*

The following easy corollary will be of use to us in Chapter 3, where computing the minimum input value will be a useful building block.

Proposition 2.23. *If $k \ge n$, then for any dynamic graph of size n and for any node u we have $\mathsf{past}(u, k - 1)_0 = V$. If all nodes cooperate to forward the smallest input value they have heard so far, then after $k - 1$ rounds all nodes have the minimum input.*

Proof. If $k \ge n$, then by Corollary 2.22 the dynamic diameter of any graph over n nodes is at most $n - 1 \le k - 1$. Therefore by Propositions 2.12 and 2.14, we have $V = \mathsf{past}(u, n - 1)_0 = \mathsf{past}(u, k - 1)_0$ and $V = \mathsf{future}(u, 0)_{n-1} = \mathsf{future}(u, 0)_{k-1}$. If in each round all nodes send the smallest value they have heard so far, $k - 1$ rounds are sufficient for the true minimum to reach all other nodes: if u is a node that has the minimum input, it is easy to show by induction on t that at each time $t \ge 0$, all nodes in $\mathsf{future}(u, 0)_t$ have u's input value. In particular at time $k - 1$ (i.e., after round $k - 1$) all nodes in $\mathsf{future}(u, 0)_{k-1} = V$ have u's input value. □

Finally, let us introduce one of the main problems studied in this thesis: *hearing from m nodes*. Informally, the problem requires nodes to analyze information flow in the network, and halt when they have been causally-influenced by sufficiently many nodes.

Definition 2.24 (Hearing from m nodes, HF_m). *In the* hearing from m nodes *task (denoted HF_m), each node u must halt at some time t such that* $|\mathsf{past}(u,t)_0| \geq \min\{m,n\}$*, where n is the size of the network. The size of the network may or may not be known in advance.*

We are especially interested in the variant HF_n, where nodes must halt only when their past-set includes the initial states of all nodes in the graph. Informally, HF_n is necessary to compute functions of the nodes' initial states, and we will see throughout this thesis that the complexity of HF_n often dominates the complexity of computing whichever function of the initial states we were truly interested in.

2.4 Epistemic Logic and Information Flow

When proving lower bounds in distributed computing, it is often useful to reason about what facts a node "knows" about the current execution — and more importantly, what facts it does *not* know. To reason formally about such knowledge we use *epistemic logic* [48]. Informally, we say that a node *knows* some fact about the current execution if its state contains sufficient information to distinguish the current execution from any execution where the fact does not hold.

Definition 2.25 (Knowledge). *Fix a set $\mathcal{C}$ of executions, an execution $\alpha \in \mathcal{C}$, and a node u participating in the execution. Let P be a Boolean predicate over executions. We say that at time t in α node u* knows *fact P (with respect to the class $\mathcal{C}$) if for any execution $\beta \in \mathcal{C}$ (including α itself), if u participates in β and α and β are indistinguishable to u at time t (that is, u's state at time t is the same in α and β), then β satisfies P.*

The class $\mathcal{C}$ will usually be the set of all executions of a specific algorithm, in which the input is drawn from the domain of the problem being studied and the dynamic graph satisfies some constraints (or often is not constrained). When the problem and the graph constraints are understood from the context, we leave $\mathcal{C}$ implicit (e.g., when we say "there exists an execution", this is to be understood as "there exists an execution in $\mathcal{C}$", and so on).

In this thesis we are especially interested in the effect of *prior information* on the complexity of solving various tasks. We say that a fact is *known a priori* if it is known to all nodes at time 0 in every execution of $\mathcal{C}$.

Example 2.26. The size of the network is known a priori in the following cases (among others):

- All nodes u receive the same number n as part of their input, and $\mathcal{C}$ contains only executions in which n equals the size of the dynamic graph. (Note that n may differ from execution to execution; it simply provides a means for nodes to distinguish between executions with different graph sizes.)
- There is some constant n such that in all executions of $\mathcal{C}$ the dynamic graph has size n. (In this case Definition 2.25 holds for the property $P =$ "the size of

the network is n" because there do not exist executions in which the property does not hold.)

Similarly, an *upper bound* on the size of the network is known a priori in the following cases:

- The two examples above (in these cases an *exact upper bound* is known),
- The set $\mathcal{U}$ from which UIDs are drawn is finite (in this case an *upper bound of* $|\mathcal{U}|$ is known),
- Each node u receives a number N_u as part of its input, and $\mathcal{C}$ contains only executions in which N_u is no smaller than the size n of the dynamic graph. If in addition there is some function $f : \mathbb{N}^+ \to \mathbb{N}^+$ such that $\mathcal{C}$ contains only executions in which $N_u \leq f(n)$ (where n is the size of the dynamic graph), then we say that *an upper bound of $f(n)$ is known.*

In Section 2.3 we claimed informally that the causal order captures all the information a node can possibly acquire about an execution; let us now make the case formally. We say that node u *cannot distinguish between executions α and β at time t*, and denote $(\alpha, t) \sim_u (\beta, t)$, if node u has the same state at time t in α and β. We now show that if two executions differ only in ways that are "visible" to nodes outside $\mathsf{past}(u, t)_0$, then node u cannot distinguish them at time t.

Definition 2.27. *Given a node u and a time t, we say that two executions α, β are (u, t)-*identical *if*

(a) Node u participates in both α and β,

(b) Node u's input is the same in α and β, and

(c) In each round up to time t, node u has the same incoming edges in α and in β.

*We say that α and β are (S, t)-*identical, *where S is a set of nodes, if α and β are (u, t)-identical for every $u \in S$.*

Proposition 2.28. *Let α be an execution with dynamic graph $G = (V, E)$, and fix $u \in V$ and a time t. Let β be an execution such that for all $t' \leq t$, α and β are $(\mathsf{past}(u, t)_{\alpha,t'}, t')$-identical: that is, for all $(v, t') \in \mathsf{past}(u, t)$, executions α, β are (v, t')-identical. Then $(\alpha, t) \sim_u (\beta, t)$.*

Proof. We show by induction on $t' \leq t$ that α and β are indistinguishable at time t' to all nodes in $\mathsf{past}(u, t)_{\alpha,t'}$. The base case is immediate: α and β are $(\mathsf{past}(u, t)_0, 0)$-identical, so all nodes of $\mathsf{past}(u, t)_{\alpha,0}$ have the same input in α and β. This means these nodes have the same initial states in α and in β, and hence they cannot distinguish the two executions.

For the induction step, assume that the executions are indistinguishable up to time t' to all nodes in $\mathsf{past}(u, t)_{\alpha,t'}$, and consider time $t' + 1 \leq t$. Fix $v \in \mathsf{past}(u, t)_{\alpha,t'+1}$. In round $t' + 1$ node v's incoming edges are the same in α, β. Moreover, if $(w, v) \in$

$E(t'+1)$ then $(w,t') \rightarrow_G (v,t'+1) \rightsquigarrow_G (u,t)$, and hence $w \in \mathsf{past}(u,t)_{\alpha,t'}$. By the induction hypothesis each such node w cannot distinguish α from β at time t', and since $(v,t') \rightsquigarrow_G (v,t'+1) \rightsquigarrow_G (u,t)$, neither can v itself. Therefore node v receives the same messages in round $t'+1$ of α and β and its state at time $t'+1$ remains the same in both.

By Proposition 2.12 we have $u \in \mathsf{past}(u,t)_{\alpha,t}$. The claim follows by applying the induction hypothesis to node u at time t. □

Informally speaking, the proposition above asserts that if $(v,t') \not\rightsquigarrow (u,t)$, then node u cannot know at time t any non-trivial fact about node v at time t', where here "non-trivial" means that the fact is not satisfied in some other execution, and "a fact about node v" is a fact that can be made false by changing only the input to node v or the edges that hit node v, or by omitting node v from the execution entirely.

Chapter 3

Counting and Information Dissemination in Dynamic Networks

In this chapter we study two basic primitives for dynamic networks: *information dissemination* and *counting*. An information dissemination task requires all nodes to collect the initial inputs of all the other nodes; this is a powerful primitive, which allows nodes to compute any global function of their initial inputs. Counting simply requires nodes to determine the size of the network, or to find an upper bound on the size (precise definitions will be introduced in the sequel). Counting and information dissemination are closely related: informally, as part of information dissemination, nodes must determine the size of the network in order to know when they have collected everyone's information. In this chapter we will show that both tasks can be solved efficiently even when no *a priori* information about the network is available to the nodes.

We begin in Section 3.1 by introducing the formal problem definitions and proving a few simple facts regarding their relationships to other problems. In Section 3.2 we give a simple counting / information dissemination algorithm that uses large messages, but demonstrates a key idea that will be used throughout. In the remainder of the chapter we study solutions for counting and information dissemination using small messages. We begin in Sections 3.3–3.4 by showing that in 1-interval connected graphs, we can solve both problems deterministically in $O(n^2)$ rounds using messages of size $O(\log n)$. In Section 3.4.2 we turn our attention to ∞-*interval connected graphs*, and give a different approach that allows us to solve counting and information dissemination in $O(n)$ rounds in such graphs. Next, combining ideas from both approaches, we show in Section 3.4.3 that counting and information dissemination can be solved deterministically in $O(n + n^2/T)$ rounds in T-interval connected graphs. We address several weaker variants of the model in Section 3.5.

The solutions above are all deterministic; in Section 3.6 we give a randomized approximate counting algorithm which terminates in $O(n)$ rounds with high probability. We conclude in Section 3.7 with an $\Omega(n + nk/T)$ lower bound on a restricted class of algorithms for exchanging k pieces of information in a T-interval connected graph.

3.1 Problem Definitions and Basic Properties

We define several variants of information dissemination and counting. The following tasks are all untimed, unless otherwise noted.

Definition 3.1 (Token Dissemination). *In the* k-token dissemination task, *each node receives an initial set of* tokens *drawn from some domain, such that the total number of tokens in the input to all nodes is k. The goal is for each node to collect and output all k tokens. The nodes may or may not know k in advance; if k is not known, part of the task is to determine k, since nodes cannot halt until they have collected all k tokens.*

Expressed as a task, the input to each node u is a set $I(u) \subseteq \mathcal{D}$, where $\mathcal{D}$ is the domain from which tokens are drawn. The input/output relation is given by

$$\left\{(I, \bigcup \text{image}(I)) \mid I \in \left(2^{\mathcal{D}}\right)^{\mathbb{N}} \wedge \left|\bigcup \text{image}(I)\right| = k\right\}.$$

All-to-all token dissemination *is a variant of k-token dissemination in which $k = n$, each node receives a unique token in its input, and the nodes do not know n in advance.*

Definition 3.2 (Counting and Approximate Counting). *In the* counting task, *nodes receive no input, and each node must output the size n of the network. To express counting as a task, we assign all nodes the constant input 1, and require all nodes to output the sum of the inputs.*

In ε-approximate counting *nodes again receive no input, and must each output an approximate count $\tilde{n} \in \mathbb{R}$, such that $|\tilde{n} - n| \leq \varepsilon \cdot n$. We do not require all nodes to output the same approximate count (although the algorithm we give in Section 3.6 does have this property). To express ε-approximate counting as a task we once again assign all nodes the fixed input 1 and require each node to output an ε-approximation to the sum of the inputs.*

Definition 3.3 (k-Verification). *In the* k-verification task, *all nodes are given a bound k in their input, and must output a Boolean value indicating whether $n \geq k$ (i.e., whether the size of the input multiset, which is the number of participants in the execution, is at least k). Unlike the previous tasks, k-verification has a timing constraint: we require* simultaneous termination, *that is, all nodes must halt at the same time.*

Clearly, all-to-all token dissemination is "the hardest" single-shot task: any other task can be solved using all-to-all token dissemination by having all nodes collect all the inputs to the other nodes and then compute the solution locally.

Proposition 3.4. *Any single-shot task* $(0, 0)$*-reduces to all-to-all token dissemination over the data domain of the task.*

Using the dynamic graph properties we saw in Section 2.3, we can point out several other relationships between the problems defined above and other useful tasks.

Proposition 3.5. *Computing the minimum input value* $((1+\varepsilon)n/(1-\varepsilon), O(\log n))$*-reduces to* ε*-approximate counting.*

Proof. If $\tilde{n}_u$ is an ε-approximate count known to node u, then $|\tilde{n}_u - n| \leq \varepsilon \cdot n$, and in particular $n \leq \tilde{n}_u/(1-\varepsilon)$. We can reduce minimum to ε-approximate counting as follows: while the ε-approximate counting algorithm is running, all nodes forward the smallest input value they have heard so far (appending $O(\log n)$ bits to the messages of the approximate counting algorithm). When the ε-approximate counting algorithm terminates at node u and outputs an approximate count $\tilde{n}_u$, node u continues running until time $\tilde{n}_u/(1-\varepsilon)$ (if the ε-approximate counting algorithm terminates before this time), still forwarding in each round the smallest input value it has heard so far. Then node u halts and outputs the smallest value it received.

In every execution where the ε-approximate counting algorithm outputs a "correct" (i.e., ε-approximate) count at all nodes, every node u forwards the smallest value it has heard for at least $\tilde{n}_u/(1-\varepsilon) \geq n$ rounds. Corollary 2.22 states that any 1-interval connected graph has a dynamic diameter of at most $n-1$. Since all nodes run for at least n rounds, we have $\mathsf{past}(u,n)_0 = V$ for all nodes u; that is, at time n all nodes have the global minimum. Since the minimum value received by node u can never increase (and also never decreases below the global minimum), all nodes eventually output the global minimum.

As for the running time of the reduction, if $\tilde{n}_u$ is an ε-approximate count then $\tilde{n}_u \leq (1+\varepsilon)n$, so the reduction requires at most $(1+\varepsilon)n/(1-\varepsilon)$ additional rounds. □

The reduction above is in some sense pessimistic: it ensures that nodes run for at least $n-1$ rounds, which in turn guarantees that if node u halts at time t then $\mathsf{past}(u,t)_0 = V$. However, in some cases the dynamic diameter of the graph can be much smaller than $n-1$, and in such cases counting and minimum can be solved in much fewer than $n-1$ rounds. (For example, in Section 4.3.3 we will see that we can detect in $O(1)$ rounds whether the graph is a clique, and if so we can solve all-to-all token dissemination in $O(1)$ rounds.) Must we always waste $n-1$ rounds, even if the counting algorithm terminates before time $n-1$? In other words, are there any situations where counting is faster than computing a minimum? It turns out that the answer is no: counting is at least as hard as hearing from everyone (HF_n), which in turn is no harder than computing a minimum. Informally speaking, the following lemma shows that if node u halts before it has heard from everyone (i.e., before its past-set from time 0 is V), then node u "knows nothing" about the size of the network – including the exact count or any non-trivial approximation of the count.

Lemma 3.6. *In the absence of an* a priori *bound on the size of the network, the* HF_n *task* $(0,0)$*-reduces to* ε*-approximate counting for any* n*.*

Proof. We claim that if $\mathcal{A}$ is a randomized algorithm for ε-approximate counting, then in any execution of $\mathcal{A}$, with high probability (or deterministically for deterministic algorithms) each node u halts at some time t such that $\mathsf{past}(u,t)_0 = V$. In other words, we will show that $\mathcal{A}$ already solves HF_n all by itself. Thus, a trivial $(0,0)$-reduction from HF_n is obtained by simply running $\mathcal{A}$ until it halts, and then halting.

Let us show that any ε-approximate counting algorithm must already solve HF_n. For clarity, we first present the proof for deterministic algorithms, and then generalize the proof to randomized algorithms.

We claim that if $\mathcal{A}$ is a deterministic algorithm for ε-approximate counting, then in any execution of $\mathcal{A}$, no node u can halt at time t unless $\mathsf{past}(u,t)_0 = V$. Suppose for the sake of contradiction that this is not the case: that is, in some dynamic graph $G = (V, E)$ with $|V| = n$, some node $u \in V$ halts at time t such that $\mathsf{past}(u,t)_0 \neq V$ and outputs a count $\tilde{n}_u$. Let $v \in V \setminus \mathsf{past}(u,t)_0$, that is, $(v,0) \not\rightsquigarrow (u,t)$. Informally, we will show that u cannot estimate the size of V at time t, because arbitrarily many nodes could be "hidden behind" node v, which node u does not hear from.

Consider an alternative execution with dynamic graph $G' = (V', E')$, where $V' = V \cup \{x_1, \ldots, x_{\lceil 2\varepsilon n/(1-\varepsilon)\rceil+1}\}$ for some fresh UIDs $x_1, \ldots, x_{\lceil 2\varepsilon n/(1-\varepsilon)\rceil+1} \notin V$.[1] We have

$$|V'| = |V| + \left\lceil \frac{2\varepsilon n}{1-\varepsilon} \right\rceil + 1 > n + \frac{2\varepsilon n}{1-\varepsilon} = \frac{1+\varepsilon}{1-\varepsilon} n. \tag{3.1.1}$$

The edges E' of G' are defined by

$$E'(r) = E(r) \cup \{(w,v),(v,w) \mid w \in V' \setminus V\},$$

that is, in each round we take all the edges of $G(r)$ and add edges between all the new nodes ($V' \setminus V$) and node v. By definition, G and G' (more precisely, the executions induced by G and G') are $(V \setminus \{v\}, t)$-identical: all nodes of $V \setminus \{v\}$ participate in G', and they have the same input (the constant 1) and the same incoming edges. Therefore Proposition 2.28 shows that node u cannot distinguish G' from G at time t; the new nodes are "hidden" from u because they are connected only to node v, which node u does not hear from. Thus, at time t in G' node u's state is the same as it is at time t in G, and hence in G' it also halts and outputs $\tilde{n}_u$. However, $\tilde{n}_u$ cannot be an ε-approximate count for both G and G': if $\tilde{n}_u$ is an ε-approximate count for G, then

$$\tilde{n}_u \leq (1+\varepsilon)n \overset{(3.1.1)}{<} (1-\varepsilon)|V'|.$$

This shows that no node can halt before it has heard from everyone, and completes the proof for deterministic algorithms.

Now suppose that $\mathcal{A}$ is a randomized algorithm for ε-approximate counting which succeeds with probability at least $p(n) = 1 - o(1)$ in networks of size n. Fix a graph $G = (V, E)$ of size n, and let B be the event that when $\mathcal{A}$ is executed in G, some node halts before hearing from everyone (i.e., there exists a node u that halts at some time t such that $\mathsf{past}(u,t)_0 \neq V$). We will show that $\Pr[B] \leq 1 - p(n) = o(1)$, that is, $\mathcal{A}$ solves HF_n with high probability.

Let us decompose B into a set of not necessarily disjoint events $\{B_{u,v} \mid u, v \in V\}$, where $B_{u,v}$ is the event that in G, node u halts before hearing from node v. For

[1] Recall that we assume no *a priori* upper bound on the size, and this means that the UID space is unbounded, otherwise the size of the UID space represents a bound on the size of the network (see Section 2.1). Therefore we can always add as many nodes as necessary.

each $v \in V$ and $N > n$, we construct a graph $H_{v,N} = (V_{v,N}, E_{v,N})$ where node u errs with high probability (note that the construction of $H_{v,N}$ is independent of u and depends only on v and N). The graph $H_{v,N}$ is similar to the graph G' we used in the deterministic case: we set $V_{v,n} := V \cup \{x_1, \dots, x_{N-n}\}$, where $x_1, \dots, x_{N-n} \notin V$, and $E_{v,N}(r) := E(r) \cup \{(x_i, v), (v, x_i) \mid 1 \le i \le N - n\}$ for all r. Let $C_{u,v,N}$ be the event that in $H_{v,N}$, node u halts before hearing from v. As we saw above for the deterministic case, if $v \notin \mathsf{past}(u,t)_{H_{v,N},0}$ (or equivalently $v \notin \mathsf{past}(u,t)_{G,0}$), then we have $\mathsf{past}(u,t)_{H_{v,N}} = \mathsf{past}(u,t)_G$, that is, node u cannot "see" the difference between G and $H_{v,N}$. Since the behavior of node u depends only on the time-nodes in its past set, and the random choices of the time-nodes in u's past set are independent of the states and randomness of time-nodes outside u's past set, $\Pr[C_{u,v,N}] = \Pr[B_{u,v}]$, and moreover, the distribution of node u's output in $H_{v,N}$ given $C_{u,v,N}$ is the same as the distribution of node u's output in G given $B_{u,v}$. We can choose N sufficiently large so that any correct ε-approximate count for G is an incorrect output in $H_{v,N}$ (choosing $N > (1+\varepsilon)n/(1-\varepsilon)$ suffices); for any such choice of N,

$$\Pr[u \text{ errs in } H_{v,N} \mid C_{u,v,N}] \ge \Pr[u \text{ succeeds in } G \mid B_{u,v}].$$

Thus,

$$\begin{aligned} 1 - p(N) &\ge \Pr[u \text{ errs in } H_{v,N}] \ge \Pr[u \text{ errs in } H_{v,N} \mid C_{u,v,N}] \Pr[C_{u,v,N}] \\ &\ge \Pr[u \text{ succeeds in } G \mid B_{u,v}] \Pr[B_{u,v}] = \Pr[u \text{ succeeds in } G \wedge B_{u,v}]. \end{aligned}$$

The first inequality follows from the fact that $\mathcal{A}$ fails in $H_{v,N}$ whenever any node errs, and in particular when node u errs. (There is some slack here, since we analyze each node individually.) Similarly, $\mathcal{A}$ succeeds in G only when all nodes succeed, and hence

$$\begin{aligned} \Pr[\mathcal{A} \text{ succeeds in } G \wedge B] &\le \sum_{u,v \in V} \Pr[\mathcal{A} \text{ succeeds in } G \wedge B_{u,v}] \\ &\le \sum_{u,v \in V} \Pr[u \text{ succeeds in } G \wedge B_{u,v}] \le n^2(1 - p(N)). \end{aligned}$$

Finally, we can write

$$\begin{aligned} 1 - p(n) &\ge \Pr[\mathcal{A} \text{ fails in } G] \ge \Pr[\mathcal{A} \text{ fails in } G \wedge B] \\ &\ge \Pr[B] - \Pr[\mathcal{A} \text{ succeeds in } G \wedge B] \ge \Pr[B] - n^2(1 - p(N)), \end{aligned}$$

that is,

$$\Pr[B] \le 1 - p(n) + n^2(1 - p(N)) = 1 - p(n) + o(1).$$

For all $\delta > 0$, we can show by increasing N sufficiently (while n remains fixed) that $\Pr[B] \le 1 - p(n) + \delta$. Therefore we must have $\Pr[B] \le 1 - p(n)$.

□

If we have an *a priori* bound N on the size of the network, we may not be able to carry out the proof above: we cannot necessarily extend the graph G by adding as many nodes as we like (doing so may violate the bound N if the size of the graph is already close to N). However, *exact* counting is always as hard as HF_n, unless an exact count is known a priori; more generally, ε-approximate counting still requires all nodes to hear from all but a $2\varepsilon/(1+\varepsilon)$-fraction of nodes in the graph, that is, it requires all nodes to hear from a $(1-\varepsilon)/(1+\varepsilon)$-fraction of nodes.

Proposition 3.7. *For any $\varepsilon \in (0,1)$, the $HF_{((1-\varepsilon)/(1+\varepsilon))n}$ task $(0,0)$-reduces to deterministic ε-approximate counting.*

Proof. We proceed as in Lemma 3.6, except that we now remove nodes from the graph instead of adding them: the UID space may be smaller than $(1+\varepsilon)n/(1-\varepsilon)$, in which case we cannot pad the graph as we did in Lemma 3.6.

Fix a graph G of size n, and suppose for the sake of contradiction that u is a node that halts at time t such that $|\mathsf{past}(u,t)_0| < (1-\varepsilon)n/(1+\varepsilon)$. Let $\tilde{n}_u \geq (1-\varepsilon)n$ be the value output by u at time t, and let $G' = (V', E')$ be the graph induced by G on $V' := \mathsf{past}(u,t)_0$. (Note that $|\mathsf{past}(u,t)_0| \geq 1$ by Proposition 2.12, so G' is well-defined.) We have

$$|V'| < \frac{1-\varepsilon}{1+\varepsilon}n \leq \frac{\tilde{n}_u}{1+\varepsilon}.$$

Therefore $\tilde{n}_u > (1+\varepsilon)|V'|$, and $\tilde{n}_u$ is an incorrect approximate count for G'. However, node u cannot distinguish G from G' up to time t, and hence it still outputs $\tilde{n}_u$ in G'. □

So far we have shown that counting (and indeed any other task) reduces to all-to-all token dissemination, and that hearing from everyone (HF_n) reduces to counting. To complete the picture, let us show that HF_n is representative of a large class of functions, which all reduce to counting. (Refer to the task classes defined in Section 2.2.)

Proposition 3.8. *For any $b \geq 1$, the HF_n task is $(0,b)$-complete for the class of duplication-insensitive single-valued tasks over inputs drawn from $\{1,\ldots,b\}$.*

Proof. Let T be a duplication-insensitive single-valued task over inputs in $\{1,\ldots,b\}$. Then the permissible output values to T depend only on the set of values present in the input assignment, not on the multiplicities of these values.

Given an HF_n algorithm $\mathcal{A}$, we can solve T by having each node append a tuple $(x_1,\ldots,x_b)$ to each of $\mathcal{A}$'s messages, where initially $x_i = 1$ iff the node received value i in its input, and subsequently $(x_1,\ldots,x_b)$ is the pointwise-OR of all tuples received so far. When $\mathcal{A}$ halts, each node's tuple $(x_1,\ldots,x_b)$ is exactly the characteristic vector of the set of inputs. Since T is duplication-insensitive, this information is sufficient for the node to compute and output a permissible output value. □

To conclude, the results in this section show that counting is a useful primitive, since it allows us to compute "easy" (duplication-insensitive) functions of the input. Lemma 3.6 shows that a counting algorithm only terminates (or w.h.p. only terminates) when the past-set of each node includes all other nodes. Therefore if we have a function that is "easy to compute but hard to know when we are done computing", like the minimum input value, we can compute it by attaching a small amount of information (in this case the smallest input heard so far) to the messages sent by the counting algorithm, and halting when the counting algorithm halts. In Section 3.6 we will see that the relationship goes both ways: computing the minimum is also useful towards randomized approximate counting. However, in the following sections we first study deterministic exact counting.

3.2 Counting and Information Dissemination using Large Messages

Throughout this chapter we are mostly concerned with algorithms that use small messages, of polylogarithmic size. However, we remark that using messages of size $O(n \log n)$, counting and information dissemination can be solved in linear time using the algorithm below. We assume here that the information being disseminated is the set of node UIDs (and in particular, to compute the count, nodes can simply output the size of the set of UIDs they have collected). For more general information dissemination, nodes can execute the same algorithm, using pairs of the form (UID, input) everywhere that UIDs are used below.

Algorithm 3.1: Information dissemination with $O(n \log n)$-bit messages: code executed by each node

```
A ← {self}
t ← 0
while |A| > t do
    send A
    receive B_1, ..., B_s from neighbors.
    A ← A ∪ B_1 ∪ ... ∪ B_s
    t ← t + 1
end
output A
```

In the algorithm, nodes collect and forward all the UIDs they have heard, and halt when they reach a time t in which they have heard no more than t UIDs. Let $A_u(t)$ denote the value of node u's local variable A at time t. Since each node forwards all the UIDs it has heard so far, at any time t we have $A_u(t) = \mathsf{past}(u,t)_0$, i.e., $A_u(t)$ stores exactly the UIDs of nodes that causally influence u. From Lemma 2.21 we have that $|A_u(t)| \geq \min\{n, t+1\}$. If node u halts at time t, then $|A_u(t)| \leq t$, and hence $|A_u(t)| = n$ and the set output by u is exactly V. As for termination, when we reach time n the halting condition is satisfied, because we always have $|A_u(t)| \leq n$.

The main idea we wish to illustrate through this algorithm is the *termination test*, i.e., the idea of deciding to terminate when $\mathsf{past}(u,t)_0 \leq t$. In this particular instance we can think of the time t as a guess for the count n. There are two ingredients here that we will re-use in later sections: first, $|A_u(t)|$ serves as an accurate count of the number of nodes that have causally influenced u by time t (i.e., for $|\mathsf{past}(u,t)_0|$), because $A_u(t)$ contains the UIDs of exactly those nodes. Second, Lemma 2.21 shows that nodes can *expect* to be causally influenced by at least $t+1$ nodes by time t, unless $t \geq n$. These two ingredients make up the termination test, in which nodes estimate the number of nodes they have been causally influenced by, and compare this number to the number of nodes they would expect to have been causally influenced by if their guess for the count (in this case t) is too small.

In the sequel we restrict our attention to algorithms that use small messages, which rules out sending all the UIDs heard so far; we will see other ways to estimate the size of $\mathsf{past}(u,t)_0$. In Section 3.4.2 we will assume that the graph is ∞-interval connected and use a pipelining algorithm to assemble this set (with some additive delay compared to the algorithm above), and in Section 3.6 we will use randomized approximate counting to estimate the size of this set in 1-interval connected graphs. In both cases we will use termination tests similar to the test above. However, to construct a deterministic solution for finite-interval connected graphs we require another key ingredient, which we introduce below: the idea of counting through repeatedly solving k-committee election.

3.3 Counting Through k-Committee Election

In this section we introduce a new problem, *k-committee*, and show that it can be used to solve counting and token dissemination in graphs that are at least 1-interval connected. The problem is defined as follows.

Definition 3.9 (k-committee). *In the* k-committee task, *each node receives as input a parameter $k \geq 1$, and all nodes must simultaneously halt and output a* committee ID *(comprising $O(\log n)$ bits). We refer to a set of nodes that output the same committee ID as a* committee. *The output must satisfy the following conditions:*

(a) The size of each committee does not exceed k, and

(b) If $k \geq n$, all nodes must be in the same committee (i.e., all nodes output the same committee ID).

The k-committee problem is *equivalent* to the k-verification problem from Definition 3.3: that is, any solution for one of these problems also allows us to solve the other. The more useful part of this equivalence is the fact that k-committee allows us to solve k-verification. In turn, k-verification lets us solve the counting problem by trying exponentially-increasing values of k, and using k-verification to check whether each value is an upper bound on the size of the network. (Repeated doubling does not yield an exact count, but rather a 2-approximate count; we will see in Section 3.4 how to improve upon this approach to obtain an exact count.)

To reduce k-verification to k-committee we use the following algorithm, which assumes that we have already solved the k-committee task and each node u has a committee ID cid_u satisfying the conditions of Definition 3.9. Assume without loss of generality that $\perp$ is a special value that is not used as a committee ID.

The k-verification protocol (see Algorithm 3.2). For k rounds, each node u broadcasts the value of cid_u and receives the values sent by its neighbors; if it receives a different committee ID from its own, or the special value $\perp$, it sets $cid_u \leftarrow \perp$ and broadcasts that value in subsequent rounds. After k rounds, all nodes u output 1 if $cid_u \neq \perp$ and 0 otherwise.

Algorithm 3.2: The k-verification protocol, where cid is the node's input

```
1  foreach $t = 1, \dots, k$ do
2      send $cid$
3      receive $c_1, \dots, c_s$ from neighbors
4      if exists $i$ such that $c_i \neq cid$ then
5          $cid \leftarrow \perp$
6      end
7  end
8  if $cid \neq \perp$ then
9      output 1
10 else
11     output 0
12 end
```

Lemma 3.10. *The k-verification task $(k, O(\log n))$-reduces to k-committee.*

Proof. We show that if we execute the k-verification protocol above from an initial state representing a solution to k-committee, then the output of all nodes is 1 if $k \geq n$ and 0 otherwise. A $(k, O(\log n))$-reduction from k-verification to k-committee is then obtained by running the k-committee algorithm until it terminates (note that all nodes halt at the same time, since k-committee guarantees simultaneity), and then running the k-verification protocol, which requires an additional k rounds.

Suppose first that $k \geq n$. In this case, from the definition of k-committee, there is only one committee in the graph; all nodes send the same committee ID, and no node ever receives a committee ID different from its own (or $\perp$). After k rounds all nodes still have their original cid, and all output 1.

Now suppose that $k < n$. We will show that each committee "shrinks by one" in each round of the k-verification protocol, until after k rounds all committees are empty and all nodes output 0. More formally, let $S_C(i)$ be the set of nodes in committee C (i.e., the nodes u that have $cid_u = C$) after i rounds of the k-verification protocol. We show that $|S_C(i)| \leq k - i$.

The base case, $i = 0$, follows from the definition of k-committee: no committee can have more than k members. For the step, assume that $|S_C(i)| \leq k - i$ but $S_C(i) \neq \emptyset$ (otherwise the claim holds trivially), and consider a cut between $S_C(i)$ and the rest of the graph in round $i + 1 \leq k$, oriented towards $S_C(i)$. By 1-interval connectivity, there is an edge in the cut, and some node $u \in S_C(i)$ receives a value that is either $\perp$ or a committee ID different from its own; this node then sets $cid_u \leftarrow \perp$ and drops out of the committee. We have $i \in S_C(i) \setminus S_C(i+1)$, and therefore $|S_C(i+1)| \leq |S_C(i)| - 1 \leq k - (i+1)$, as desired.

After k rounds we have $S_C(k) = \emptyset$ for all committee IDs C. Therefore all nodes must have $cid = \perp$, and all output 0. □

For the sake of completeness we also prove the other direction of the equivalence: k-committee reduces to k-verification. This direction serves to show that we do not introduce any "extra difficulty" by using the k-committee problem to solve k-verification, because k-verification is already as hard as k-committee.

Lemma 3.11. *The k-committee problem $(k-1, O(\log n))$-reduces to k-verification.*

Proof. Suppose we start in a global state that represents a solution to k-verification: that is, each node already knows whether $k \geq n$ or not. Then we can solve k-committee as follows:

- If $k < n$, each node outputs its own UID as its committee ID. This is a valid solution to k-committee, because the size of each committee does not exceed k, and the second condition in Definition 3.9 does not apply (as $k < n$).
- If $k \geq n$, all nodes forward the smallest UID they have heard so far for $k-1$ rounds. After $k-1$ rounds all nodes output the smallest UID they have heard as their committee ID. Correctness follows from Proposition 2.23: since $k \geq n$, after $k-1$ rounds all nodes have the true smallest UID in the graph and only one committee is formed.

□

Remark 3.12. The fact that the communication graph is 1-interval connected was used in the k-verification protocol to guarantee that in every round, every directed cut in the graph contains at least one edge. The algorithm extends easily to the case where the graph is not 1-interval connected, but there is some constant upper bound on the number of rounds until an edge appears in any cut, and this bound is known to the nodes. Similarly, if there is some constant bound D (independent of n) on the dynamic diameter, and this bound is known to the nodes, then we can simply run the protocol for D rounds instead of $k-1$; if there is more than one committee in the graph, we are guaranteed that any node is causally influenced by some node in a different committee, so it receives either the UID of that committee or $\perp$ and eventually outputs 0.

It is not hard to see that for the purpose of counting, these two generalizations are essentially the best that we can do. If the best upper bound on the diameter known to the nodes is a function $D(n) = \omega(1)$ of the true count n, and the best upper bound on the number of rounds that can pass until an edge appears in any cut is $f(n) = \omega(1)$, then counting is impossible: for sufficiently large n, the adversary can create a partition in the graph and maintain it until all nodes halt and output an incorrect count.

3.4 Deterministic Counting and All-to-All Token Dissemination

In the previous section we showed that k-committee can be used to obtain an upper bound on the size of the network; the k-verification protocol requires only k additional rounds beyond the time required to solve k-committee. When we use exponentially-increasing guesses $k = 1, 2, 4, 8, \ldots$ the largest value we might reach before finding a value $k \geq n$ is $k = 2n$; after this point we can narrow in on the exact count using binary search, requiring an additional $O(\log n)$ rounds. Thus, if we can solve k-committee in $t(k)$ rounds, then we can solve counting in $O(t(n) + n)$ rounds by combining k-committee and k-verification.

In this section we show how to solve the k-committee task, and obtain a solution to all-to-all token dissemination (with unknown count) along the way. This renders the binary search unnecessary: while solving k-committee election, we can have each node collect the UIDs of all other nodes, so that by the time a value of $k \geq n$ is reached, all nodes have the UIDs of all other nodes and can compute the exact count.

For the sake of exposition we begin with some special cases which illustrate the main ideas. In Section 3.4.1 we give a general solution that works in 1-interval connected graphs, and requires $O(n^2)$ rounds. In Section 3.4.2 we consider the other extreme of the spectrum, ∞-interval connected graphs, and show that that all-to-all token dissemination can be solved in $O(n)$ rounds in ∞-interval connected networks. For this purpose we do not require k-committee; instead we use an approach quite similar to the large-message algorithm from Section 3.2. Finally, in Section 3.4.3 we combine the ideas used in Sections 3.4.1 and 3.4.2 to show that k-committee can be solved in $O(n + n \cdot d(\mathrm{g}, n)/T)$ rounds in (T, g)-interval connected graphs, where $d(\mathrm{g}, n)$ is the smallest integer such that $\mathrm{g}^{(d(\mathrm{g},n))}(1) > n/2$.

3.4.1 k-Committee in 1-Interval Connected Graphs

We saw in Proposition 2.23 that if $k \geq n$, then $k - 1$ rounds are sufficient for one piece of information to propagate throughout the network: the minimum input value, or similarly, any piece of information that all nodes cooperate in disseminating (by sending only this piece of information once they receive it). Our strategy for solving k-committee is to repeatedly rely on Proposition 2.23: whenever nodes wish to globally disseminate some piece of information, they forward it for $k - 1$ rounds. Using k in this way, the nodes attempt to partition themselves into committees comprising k nodes each; each committee is created by a single *leader* node, which itself belongs to the committee, and this leader issues *invitations* for exactly $k - 1$ other nodes to join the committee. Nodes join a committee only when they receive an invitation from its leader. The ultimate effect is the following.

- If $k \geq n$ then committee formation succeeds: $k - 1$ rounds are sufficient for information to reach everywhere in the network, all invitations reach their destinations, and we end up with a unique leader who successfully invites all other nodes to join its committee.

- If $k < n$ then committee formation may fail: invitations may not reach their destinations and the committees formed may be very small. However, by definition of k-committee, all we have to achieve in this case is that *no more than* k nodes join any committee, and this is guaranteed by the fact that no leader issues more than k invitations (including one to itself).

The protocol itself has two parts.

(1) Leader election: for $k-1$ rounds, all nodes in the network propagate the UID of the smallest node they have heard so far. At the end of the $k-1$ rounds, any node that has not heard a smaller UID than its own selects itself to be a leader. Leaders immediately join their own committees.

(2) Committee formation: the nodes iterate through $k-1$ cycles. Each cycle proceeds as follows.

  (a) Polling phase: for $k-1$ rounds, all nodes propagate the UID of the smallest node they have heard about that has not yet joined a committee.

  (b) After polling ends, each leader selects the smallest UID it heard during polling, and issues an invitation to that node. The invitation is a message containing the UID of the leader and that of the invited node.

  (c) Invitation phase: for $k-1$ rounds, all nodes propagate the invitations they hear. In each round each node sends no more than one invitation; if a node receives more than one invitation, it may select an arbitrary one of them to forward, or it may simply drop all invitations (the algorithm's correctness is preserved regardless of this choice).

  (d) After the invitation phase ends, invited nodes join the committee of the leader that invited them. If a node receives more than one invitation (including invitations from past cycles), it may select an arbitrary invitation to accept, or it may elect not to join a committee; the algorithm's correctness does not depend on the nodes' behavior in this case, as long as nodes do not join committees uninvited.

Finally, each node that has joined a committee outputs that committee's leader as its committee ID, and nodes that have not joined a committee output their own UID as their committee ID. Pseudocode for the algorithm is given in Algorithm 3.3. (In the pseudocode we treat $\perp$ as a special value that is smaller than all other values.)

Lemma 3.13. *The protocol above solves the k-committee problem in $O(k^2)$ rounds.*

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

(a) Each committee comprises at most k nodes: in each cycle, each node invites at most one node to join its committee. After $k-1$ cycles at most k nodes have joined a given committee (the leader of the committee and at most $k-1$ other nodes).

Algorithm 3.3: k-committee in 1-interval connected graphs

```
leader := true
cid := ⊥
// Leader election
min_uid := self
for $r = 1, \dots, k-1$ do
    send min_uid
    receive $u_1, \dots, u_s$ from neighbors
    min_uid := min {min_uid, $u_1, \dots, u_s$}
if min_uid < self then
    leader := false
else cid := self
// Committee formation
for $i = 1, \dots, k-1$ do
    // Polling phase
    if cid = ⊥ then
        candidate := self
    else
        candidate := ⊥
    for $r = 1, \dots, k-1$ do
        send candidate
        receive $u_1, \dots, u_s$ from neighbors
        candidate := min {candidate, $u_1, \dots, u_s$}
    if leader then
        invitation := (self, candidate)
    else
        invitation := ⊥
    // Invitation phase
    for $r = 1, \dots, k-1$ do
        send invitation
        receive $inv_1, \dots, inv_s$ from neighbors
        // Choose the first invitation to forward (this choice is
           completely arbitrary)
        invitation := $inv_1$
    // Invited nodes join committees
    if invitation = ($u_\ell$, self) for some $u_\ell$ then
        cid := $u_\ell$
if cid = ⊥ then
    cid := self
output cid
```

If a node is not invited in any cycle and eventually forms its own committee, then the node is not a leader, and therefore it is the only node in its committee (it did not issue any invitations). Therefore the size of all committees does not exceed k.

(b) If $k \geq n$, all nodes join the same committee: suppose that $k \geq n$, and let u be the node with the smallest ID in the network. During the leader election part, $k - 1$ rounds are sufficient for all nodes to hear about u (by Proposition 2.23), and hence u will be the only leader in the network. Next, during committee formation, $k-1 \geq n-1$ cycles are sufficient for u invite all the other nodes in the network; in each cycle, $k - 1$ rounds are sufficient for u to successfully identify the smallest uninvited node and to invite that node to join the committee (by Proposition 2.23 in both cases). Therefore all nodes join u's committee and there is only one committee in the network.

□

The protocol above is easily modified so that in the case where $k \geq n$, it also solves all-to-all token dissemination. Let t_u be the token node u received in its input. We now have nodes attach their tokens to their UIDs, 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 $(leader, (u, t_u))$. Whenever we take a minimum in Algorithm 3.3, we disregard the token and use only the UID. However, nodes record the tokens attached to all messages they receive.

When $k \geq n$, for each node in the network there are $n - 1$ rounds where the pair (u, t_u) is forwarded by all nodes in the network: for the leader these rounds occur in the leader election phase, and for the remaining nodes, when the node is invited to join the committee. Thus, if nodes store the tokens attached to UIDs they hear, then all nodes collect all tokens by the time the algorithm completes.

Proposition 3.14. *When Algorithm 3.3 is executed with $k \geq n$, and all nodes attach their tokens to their UIDs and collect all tokens they receive, all nodes collect all tokens by the time the algorithm completes.*

Combining the results above yields an $O(k^2)$-round algorithm for k-verification, which we can use to obtain an $O(n^2)$-round algorithm for exact counting and all-to-all token dissemination.

Theorem 3.15. *Exact counting and all-to-all token dissemination can be solved in $O(n^2)$ rounds in 1-interval connected networks.*

Proof. Combining Lemma 3.13 with Lemma 3.10 we obtain an $O(k^2)$-round algorithm for k-verification, with the property that all nodes halt at the same time (since k-verification and k-committee both require simultaneous termination). To solve exact counting and all-to-all token dissemination, we solve k-verification for $k = 1, 2, 4, \ldots$, attaching input tokens (or node UIDs if we are only interested in counting) to messages as in Proposition 3.14 above. When k-verification returns 1 (i.e., we reach a value of

k at least as large as the size of the network), all nodes halt, and output the set of tokens they have collected (to solve token dissemination) or the size of this set (to solve exact counting). The running time is dominated by the time required for the last iteration of k-verification, which is $O(n^2)$ rounds because $k \leq 2n$. $\square$

Remark 3.16. The counting algorithm we constructed has the property that all nodes halt at the same time. This is not an inherent requirement of the counting problem, but in some sense it comes "free of charge": if we are given a non-simultaneous exact counting algorithm with worst-case running time $t(n)$, we can turn it into a simultaneous algorithm by having all nodes execute the original algorithm but wait until time $t(n)$ before halting. Knowing an exact count is sufficient to compute $t(n)$, and the worst-case running time is unaffected by this change. We will see in Chapter 4 that counting and simultaneous termination are also closely related in another sense: any algorithm for solving a non-trivial simultaneous task must implicitly compute an upper bound on the size of the network.

Remark 3.17. So far in this section we did not assume a non-trivial bound on the dynamic diameter, beyond the bound of $n-1$ which holds for any 1-interval connected graph. It bears noting that the algorithm's running time can be reduced to $O(kD)$ if a bound of D on the dynamic diameter is known to all nodes, by simply reducing the duration of each propagation phase to D rounds instead of $k-1$. In fact, this holds even if the graph is not 1-interval connected but has a dynamic diameter of D. In general, if nodes know that the dynamic diameter of the graph is bounded by some function $f(n)$ of the count n, then they can propagate messages for $f(k)$ rounds in Algorithm 3.3, yielding a correct k-committee election algorithm that runs in $O(kf(k))$ rounds. Unfortunately, as we saw in Remark 3.12, the k-verification protocol does not extend as nicely. Therefore to solve the counting problem we still require a fixed upper bound (independent of n) on either the dynamic diameter or the number of rounds that can pass until an edge appears in any cut, and it is easy to prove that such a bound is necessary.

3.4.2 All-to-All Token Dissemination in ∞-Interval Connected Graphs

We now turn to studying ∞-interval connected graphs, as a step towards general T-interval connected graphs (Section 3.4.3). An ∞-interval connected graph $G = (V, E)$ is a dynamic graph where $\bigcap_{r \geq 0} E(r)$ induces a strongly-connected graph on V; in other words, there is a stable strongly-connected spanning subgraph $G' = (V, E')$, such that $E' \subseteq E(r)$ for all r. Along the edges of G' we enjoy a neat pipelining effect: if nodes collect all the tokens they receive and broadcast each token just once, all nodes are guaranteed to receive all tokens in $2n$ rounds.[2] This property gives rise to Algorithm 3.4 for solving the all-to-all token dissemination problem with unknown n.

[2]A similar pipelining effect was shown for static graphs in [139]; our analysis here is different and more precise. Pipelining arguments were also used in [93] to analyze multi-message global broadcast in terms of the progress bounds guaranteed by a local broadcast algorithm.

We assume that the input to node u is stored in $A_u(0)$, that is, in u's local variable A at time 0.

Algorithm 3.4: All-to-all token dissemination in ∞-interval connected graphs

```
$A \leftarrow$ {input token} // Tokens received so far
$S \leftarrow \emptyset$ // Tokens already sent
$t \leftarrow 0$ // The current time
while $|A| \geq \lfloor t/2 \rfloor$ do
    if $S \neq A$ then
        $x \leftarrow \min(A \setminus S)$
        send $x$
        $S \leftarrow S \cup \{x\}$
    end
    receive $x_1, \ldots, x_s$ from neighbors
    $A \leftarrow A \cup \{x_1, \ldots, x_s\}$
    $t \leftarrow t + 1$
end
output $A$
```

We note that for our purpose here, it is sufficient to have a node send *any* token it has not sent yet, not necessarily the smallest one. However, we will generalize this approach to finite-interval connected graphs in Section 3.4.3, and there choosing the smallest token will be significant.

For a node $u \in V$ and a token $x \in \bigcup_{v \in V} A_v(0)$, let $\text{tdist}(x, u)$ denote the shortest-path distance in the stable graph G' from any node $v \in V$ such that $x \in A_v(0)$ to node u. (In all-to-all token dissemination there is exactly one node v such that $x \in A_v(0)$; however, we will later extend this approach to a more general case where nodes can receive more than one token in their input.) The correctness of Algorithm 3.4 hinges on the following property, which shows that nodes are continually making progress. Informally, we assert that if $t \geq \text{tdist}(x, u)$, then t rounds are "enough time" for u to receive x; if u has not received x and sent it on, the path between u and the nearest node that knows x must have been blocked by other tokens, which node u received and sent on. Note that this statement concerns the tokens *sent* by node u, rather than the tokens *received* by node u; this is essential for the proof to go through. On a very high level, pipelining works because each token can impede the progress of each other token at most once. A token that u has already sent will never again block the progress of another token at node u, because nodes never send the same token twice; in contrast, tokens that u has received but not yet sent will be sent by u in some future round, delaying other tokens. Therefore, to bound the total delay a token may suffer as it propagates along some path, we must reason about the tokens the nodes on the path have already sent, not just the tokens they receive.

The following lemma is stated in a slightly more precise form than we require for this section; we will re-use it in Section 3.4.3, and the additional details will be useful there.

Lemma 3.18. *For any node $u \in V$, token $x \in \bigcup_{v \in V} A_v(0)$ and time $t \geq \text{tdist}(x, u)$, either $x \in S_u(t)$, or $S_u(t)$ includes at least $(t - \text{tdist}(x, u))$ tokens that are smaller* *than x.*

Proof. The proof is by induction on t. The base case, $t = 0$, is immediate. For the inductive step, suppose that the claim holds at time t with respect to all nodes and all tokens, and fix a node u and a token x such that $t+1 \geq \text{tdist}(x, u)$. We must show that either $x \in S_u(t+1)$ or $S_u(t+1)$ contains at least $(t+1-\text{tdist}(x, u))$ tokens smaller than x.

Let us first dispense with two easy cases. If $\text{tdist}(x, u) = 0$, then $x \in A_u(0)$, and x is in $A_u \setminus S_u$ until it is sent and added to S_u. Thus, either $x \in S_u(t+1)$, or in rounds $1, \ldots, t+1$ node u was busy sending smaller tokens than x, and these tokens were then added to S_u. The claim holds in both cases.

Next, consider the case where $t+1 = \text{tdist}(x, u)$. In this case $(t+1-\text{tdist}(x, u)) = 0$, so the claim holds trivially.

In the sequel we therefore assume that $\text{tdist}(x, u) > 0$ and that $t+1 > \text{tdist}(x, u)$, that is, $t \geq \text{tdist}(x, u)$. Let v be an in-neighbor of u with $\text{tdist}(x, v) = \text{tdist}(x, u) - 1$. (That is, v is the node before u on a shortest path from the nearest node that has x in its input to node u; distances are always measured with respect to the stable graph G'.)

We now apply the induction hypothesis twice. Since $t \geq \text{tdist}(x, u)$, we can apply the induction hypothesis at node u and time t to obtain

($\star$) Either $x \in S_u(t)$ or $S_u(t)$ contains $(t - \text{tdist}(x, u))$ tokens smaller than x.

At node v we have also have $\text{tdist}(x, v) = \text{tdist}(x, u) - 1 \leq t$ (recall that, by the conditions of the claim, $t+1 \geq \text{tdist}(x, u)$). Applying the induction hypothesis at time t shows that either $x \in S_v(t)$, or $S_v(t)$ contains $(t - \text{tdist}(x, v)) = (t+1-\text{tdist}(x, u))$ tokens smaller than x. Moreover, since v is an in-neighbor of u in G', any token sent by v is received by u in the same round and added to A_u; thus we have

($\star\star$) .Either $x \in A_u(t)$ or $A_u(t)$ contains $(t+1-\text{tdist}(x, u))$ tokens smaller than x.

We now have everything we need to prove the claim. Since u never discards tokens from S_u, we have $S_u(t) \subseteq S_u(t+1)$. If $S_u(t)$ contains x, then so does $S_u(t+1)$, and we are done. Otherwise we know from ($\star$) that $S_u(t)$ contains *at least* $(t - \text{tdist}(x, u))$ tokens smaller than x. If $S_u(t)$ contains *more* than $(t - \text{tdist}(x, u))$ tokens smaller than x, then it contains at least $(t+1-\text{tdist}(x, u))$ tokens smaller than x, and we are also done. The only remaining case is that $S_u(t)$ contains *exactly* $(t - \text{tdist}(x, u))$ tokens smaller than x, and does not contain x itself. In this case it is sufficient to show that $\min(A_u(t) \setminus S_u(t)) \leq x$: this ensures that node u sends either x or some token smaller than x in round $t+1$, and this token is then added to $S_u(t+1)$; so we either have $x \in S_u(t+1)$ or $S_u(t+1)$ contains $(t+1-\text{tdist}(x, u))$ tokens smaller than x.

Thus, let us show that $\min(A_u(t) \setminus S_u(t)) \leq x$. If $x \in A_u(t)$ then this holds, because we assumed that $x \notin S_u(t)$. If $x \notin A_u(t)$, then ($\star\star$) shows that $A_u(t)$ contains $(t+1-\text{tdist}(x, u))$ tokens smaller than x, and we assumed that $S_u(t)$ contains only $(t - \text{tdist}(x, u))$ such tokens; therefore $A_u(t) \setminus S_u(t)$ contains at least one token smaller than x, and we are done. □

Corollary 3.19. *For all nodes u and for all $t \leq n$ we have $|A_u(2t)| \geq t$.*

Proof. In all-to-all token dissemination each node receives a unique token in its input. Since G' is strongly-connected, the t-in-neighborhood of u in G' contains at least t nodes, each with a different input; consequently there are at least t different tokens x that satisfy $t \geq \text{tdist}(x, u)$. Applying Lemma 3.18 we see that for each token x with $t \geq \text{tdist}(x, u)$, either $x \in A_u(2t)$ or $A_u(2t)$ contains $(2t - \text{tdist}(x, u)) \geq t$ tokens smaller than x. Since there are at least t such tokens x, it follows that $|A_u(2t)| \geq t$. □

Theorem 3.20. *Algorithm 3.4 solves the all-to-all token dissemination problem, and consequently also counting, in $2n + 2$ rounds.*

Proof. Suppose that node u halts at time t. Then $|A_u(t)| < \lfloor t/2 \rfloor$, and since $2\lfloor t/2 \rfloor \leq t$ and the size of A_u is non-decreasing (elements are never removed), we also have $|A_u(2\lfloor t/2 \rfloor)| \leq |A_u(t)| < \lfloor t/2 \rfloor$. Corollary 3.19 shows that we must have $\lfloor t/2 \rfloor > n$ and therefore $t \geq 2n$. From strong connectivity of G', for any token x in the input we have $\text{tdist}(x, u) \leq n - 1$, and hence $t - \text{tdist}(x, u) \geq 2n - (n - 1) = 2n - n + 1 = n + 1$. Lemma 3.18 shows that $x \in S_u(t)$, as there do not exist $t - \text{tdist}(u, x) \geq n + 1$ tokens smaller than x. This shows that the set output by u at time t is the set of all tokens.

As for termination, the size of $A_u(t)$ never exceeds n (there are only n tokens in the input), and hence at time $2n + 2$ we have $\lfloor (2n + 2)/2 \rfloor = n + 1 > |A_u(2n + 2)|$ and all nodes halt. □

3.4.3 Solving Counting and Token Dissemination in T-Interval Connected Graphs

In Sections 3.3 and 3.4.1 we developed the basic approach of counting by k-committee election, and in Section 3.4.2 we showed that in more stable graphs we can use pipelining to achieve faster information dissemination. We now combine these ingredients to obtain fast counting and token dissemination in T-interval connected graphs for finite T. We also bring in the vertex growth of the graph, which we showed in Section 2.3 can lead to a smaller dynamic diameter.

The k-verification protocol we gave in Section 3.3 requires only k rounds, while the k-committee election protocol from Section 3.4.1 requires $O(k^2)$ rounds and forms the bottleneck in the counting algorithm. Thus, our goal is to speed up k-committee election. The basic idea is to operate in "batches" of nodes: instead of finding and inviting nodes one by one, committee leaders will now find and invite nodes in batches of $\Theta(T)$ nodes each. Using the fact that the graph is T-interval connected, we employ pipelining to disseminate $\Theta(T)$ UIDs in $O(n)$ time, rather than just one UID as in Section 3.4.1. Finally, if the graph enjoys good vertex growth, information dissemination requires fewer rounds, as each node quickly acquires all the tokens in its neighborhood.

For convenience, we assume from now on that the graph is $2T$-interval connected, and we begin by showing how to disseminate T pieces of information in $O(n)$ rounds. These results concern only undirected graphs, because we rely on bidirectional neighborhood growth, as in Lemma 2.20. However, if one omits the portions relating to non-trivial vertex growth and assumes only the trivial vertex growth of 1, the results are easily shown to apply to directed graphs as well. In the sequel we assume that the graph is undirected, unless stated otherwise.

Disseminating T Tokens Quickly in $2T$-Interval Connected Graphs

Suppose that nodes initially receive an arbitrary number of tokens, but we wish to disseminate only the T smallest tokens in the input to all nodes in the graph. (This problem is similar to T-token dissemination, except that we do not require the input assignment to contain only T tokens in total.) We begin by executing the first $2T$ rounds of Algorithm 3.4. Although Lemma 3.21 no longer applies in its general form, we can still use it to reason about the first $2T$ rounds with respect to the stable subgraph that exists throughout those rounds. Thus we can show that each of the T smallest tokens is acquired by every node at distance at most T from some node that knows the token initially (at least T nodes, but possibly more, depending on the vertex growth of the graph). After these first $2T$ rounds, we "reset" the algorithm by clearing the set S of tokens already sent, and execute another $2T$ rounds of Algorithm 3.4.

To know (or rather, guess) when we have repeated this process enough times we use a guess $\tilde{D}$; if the graph is $(2T, \mathrm{g})$-interval connected and contains n nodes, then a "correct" value for $\tilde{D}$ is one that satisfies $\tilde{D} \geq 2d(\mathrm{g}, n)$, where

$$d(\mathrm{g}, n) := \min \left\{ d \in \mathbb{N} \mid \mathrm{g}^{(d)}(1) > n/2 \right\}.$$

Each time we execute $2T$ rounds of Algorithm 3.4, each of the T smallest tokens spreads to the entire T-neighborhood of all nodes that know it at the beginning of the $2T$ rounds. It is sufficient to repeat this process roughly $\tilde{D}/T$ times to ensure that all nodes learn each of the T smallest tokens.

In Lemma 2.20 we showed that $2d(\mathrm{g}, n)$ is an upper bound on the dynamic diameter of the graph. The argument we use here will be quite similar, except that we are interested only in edges that belong to the stable graphs that $2T$-interval connectivity implies, because the pipelining effect is only guaranteed to occur along stable paths. This is why we cannot use the *actual* dynamic diameter of the graph (which could be much smaller than $2d(\mathrm{g}, n)$) as a bound on the time required for token dissemination: a small dynamic diameter could result from many transient edges, but although these edges can never hurt, we cannot rely on them for pipelining.

The algorithm is formalized in Algorithm `disseminate`, which takes three arguments: a set of tokens A (the node's input), the stability parameter T, and a guess $\tilde{D}$. The algorithm is only guaranteed to succeed if the graph is $(2T, \mathrm{g})$-interval connected and $\tilde{D} \geq 2d(\mathrm{g}, n)$; in that case, the T smallest tokens are disseminated to all nodes. In particular, if there are only T tokens (i.e., if we are solving true T-token dissemination), then all nodes acquire all tokens.

We refer to each iteration of the outermost loop as a *phase*. Because the graph is $2T$-interval connected and each phase lasts exactly $2T$ rounds, in each phase i there is a stable connected subgraph G_i that persists throughout the phase. If there is more than one such graph, we choose G_i to be one among them that has the largest vertex

Algorithm 3.5: disseminate($A, T, \tilde{D}$)

```
1  $A \leftarrow$ {input tokens} // Tokens received so far
2  $S \leftarrow \emptyset$ // Tokens already sent
3  for $i = 0, \ldots, 2\lceil \tilde{D}/(2T) \rceil - 1$ do
4      for $r = 1, \ldots, 2T$ do
5          if $S \neq A$ then
6              $x \leftarrow \min(A \setminus S)$
7              broadcast $x$
8              $S \leftarrow S \cup \{x\}$
9          end
10         receive $x_1, \ldots, x_s$ from neighbors
11         $A \leftarrow A \cup \{x_1, \ldots, x_s\}$
12     end
13     $S \leftarrow \emptyset$
14 end
15 return $A$
```

growth, so that if the dynamic graph is $(2T, \mathrm{g})$-interval connected, then each G_i has vertex growth g.

For convenience we use a pair (i, t) to denote time $i \cdot 2T + t$, that is, the time immediately following the t-th round in phase i, where $0 \leq t \leq 2T$. Note that times $(i, 2T)$ and $(i+1, 0)$ are actually the same time $(i+1) \cdot 2T$, which is both the end of phase i and the beginning of phase $i+1$. This will be convenient when we argue by induction on the number of phases executed.

We say that u *knows* token x whenever $x \in A_u$. Let $K_i(x)$ denote the set of nodes that know x at the beginning of phase i (i.e., at time $(i, 0)$), and let $\mathrm{tdist}_i(x, u)$ denote the distance in G_i from any node in $K_i(x)$ to u. The key property of Algorithm 3.5 is the following.

Lemma 3.21. *For any node $u \in V$, token $x \in \bigcup_{v \in V} A_v(0)$ and time (i, t) such that $\mathrm{tdist}_i(x, u) \leq t \leq 2T$, either $x \in S_u(i, t)$ or $S_u(i, t)$ includes at least $(t - \mathrm{tdist}_i(x, u))$ tokens smaller than x.*

The proof is identical to the proof of Lemma 3.18, except that the induction is carried through only $2T$ rounds; we do not repeat the proof here. We now use the lemma to show that in each phase, each of the T smallest tokens spreads to the entire T-neighborhood of the nodes that know it at the beginning of the phase. Recall from Section 1.6 that $\Gamma_G^{(d)}(S)$ denotes the set of nodes at distance at most d from some node in S in graph G.

Corollary 3.22. *For any token $x \in \bigcup_{v \in V} A_v(0)$ of the T smallest tokens and for any phase i, we have $\Gamma_{G_i}^{(T)}(K_i(x)) \subseteq K_{i+1}(x)$.*

Proof. Fix a token x among the T smallest tokens and a node $u \in \Gamma_{G_i}^{(T)}(K_i(x))$, i.e., a node u such that $\mathrm{tdist}_i(x, u) \leq T$. Lemma 3.21 shows that at time $(i, 2T)$, either

$x \in S_u(i,t)$ or $S_u(i,t)$ includes at least $2T - T = T$ tokens smaller than x; the latter case is impossible because x is one of the T smallest tokens. Therefore $x \in S_u(i,t)$, which implies that u knows x; that is, $u \in A_u(i,t)$, or equivalently $u \in K_{i+1}(t)$. □

Since each stable graph G_i has vertex growth g, it is easy to show the following:

Lemma 3.23. *For each i and for any set $S \subseteq V$ we have*

$$|\Gamma_{G_i}^{(T)}(S)| \geq \min\left\{\mathrm{g}^{(T)}(|S|), n/2+1\right\}.$$

Proof. This is simply Lemma 2.18, instantiated with $d = T$. □

Now we can show that after $2\lceil \tilde{D}/(2T)\rceil$ phases, each of the T smallest tokens is successfully disseminated to all nodes in the graph.

Lemma 3.24. *If x is one of the T smallest tokens in the input, the graph is undirected and $(2T, \mathrm{g})$-interval connected, and $\tilde{D} \geq 2d(\mathrm{g}, n)$, then at the end of Algorithm* **disseminate**$(A, T, \tilde{D})$ *all nodes know token x.*

Proof. For convenience, let $p := \lceil \tilde{D}/(2T)\rceil$ denote half the number of phases Algorithm `disseminate` uses. We first consider only the first p phases, and show by induction on the phase number $i = 0, \ldots, p$ that

$$|K_i(x)| \geq \min\left\{\mathrm{g}^{(i\cdot T)}(1), n/2+1\right\}. \tag{3.4.1}$$

This shows that the token makes quick progress in the first phases, as the set of nodes that know it grows quickly. The base case is immediate, and the induction step follows from Corollary 3.22, which shows that $\Gamma_{G_i}^{(T)}(K_i(x)) \subseteq K_{i+1}(x)$, and from Lemma 3.23, which shows that $|\Gamma_{G_i}^{(T)}(K_i(x))| \geq \min\left\{\mathrm{g}^{(T)}(|K_i(x)|), n/2+1\right\}$.

Now we turn our attention to the last p phases, and show that the token still makes quick progress, as the set of nodes that do not know it shrinks quickly. Let $\bar{K}_i(x) := V \setminus K_i(x)$ denote the set of nodes that do not know x at the beginning of phase i. Using a similar induction to the one above, except going back instead of forward, we can show that for each $i = 2p, \ldots, p$,

$$|\bar{K}_i(x)| \geq \min\left\{\mathrm{g}^{((2p-i)\cdot T)}(|\bar{K}_{2p}(x)|), n/2+1\right\}. \tag{3.4.2}$$

The base case is once again immediate. For the inductive step, we rely on Lemma 3.23 to show that $|\Gamma_{G_i}^{(T)}(\bar{K}_i(x))| \geq \min\left\{\mathrm{g}^{(T)}(|\bar{K}_{i+1}(x)|), n/2\right\}$, and on Corollary 3.22 to show that $\Gamma_{G_i}^{(T)}(\bar{K}_{i+1}(x)) \subseteq \bar{K}_i(x)$. The last step is, informally speaking, the contrapositive of Corollary 3.22, which asserts that $\Gamma_{G_i}^{(T)}(K_i(x)) \subseteq K_{i+1}(x)$: if $u \in \Gamma_{G_i}^{(T)}(\bar{K}_{i+1}(x))$ then there is some node $v \in \bar{K}_{i+1}(x)$ such that $\mathrm{dist}_{G_i}(v,u) \leq T$, and Corollary 3.22 shows that we cannot have $u \in K_i(t)$, because then we would also have $v \in K_{i+1}(x)$ (the graph is undirected, so $\mathrm{dist}_{G_i}(u,v) = \mathrm{dist}_{G_i}(v,u)$).

Let us now put the two pieces together. Suppose for the sake of contradiction that $|\bar{K}_{2p}(x)| \geq 1$, that is, some node does not know x at the end of the last phase. By assumption, $\tilde{D} \geq 2d(\mathrm{g}, n)$, and therefore

$$p \cdot T = \lceil \frac{\tilde{D}}{2T} \rceil \cdot T \geq \frac{\tilde{D}}{2} \geq d(\mathrm{g}, n),$$

and thus by definition of $d(\mathrm{g}, n)$ we have $\mathrm{g}^{(p \cdot T)}(1) > n/2$. For phase p, (3.4.1) yields

$$|K_p(x)| \geq \min \left\{ \mathrm{g}^{(p \cdot T)}(1), n/2 + 1 \right\} > n/2$$

and (3.4.2) yields

$$|\bar{K}_p(x)| \geq \min \left\{ \mathrm{g}^{(p \cdot T)}(|\bar{K}_{2p}(x)|), n/2 + 1 \right\} \geq \min \left\{ \mathrm{g}^{(p \cdot T)}(1), n/2 + 1 \right\} > n/2.$$

But $K_p(x)$ and $\bar{K}_p(x)$ are disjoint by definition, so this is a contradiction. □

An easy modification of the proof above shows that for directed graphs we still enjoy fast dissemination, albeit without taking into account the vertex growth.

Lemma 3.25. *If x is one of the T smallest tokens in the input, the graph is $2T$-interval connected, and $\tilde{D} \geq n$, then at the end of* ***disseminate***$(A, T, \tilde{D})$ *all nodes know token x.*

Proof. As in Lemma 3.24, except that we use only the forward induction, carry it up to the end of the last phase ($2p$), and instead of Lemma 3.23 we rely on the fact that $|\Gamma_{G_i}^{(T)}(S)| \geq \min \{|S| + T, n\}$ for all sets $S \subseteq V$, which follows from strong connectivity of G_i. □

Counting and Token Dissemination

To solve counting and token dissemination with up to n tokens (where n is unknown), we use Algorithm `disseminate` to speed up the k-committee protocol from Section 3.4. Instead of inviting one node in each cycle, we can use `disseminate` to have the leader learn the UIDs of the T smallest nodes in the polling phase, and use Algorithm `disseminate` again to extend invitations to all T smallest nodes in the invitation phase. Thus, in $O(\tilde{D}+T)$ rounds we can increase the size of the committee by T.

Recall that $\tilde{D}$ is the nodes' guess for $2d(\mathrm{g}, n)$. This guess should of course depend on the guess k for the count: if the network is known to be $(2T, \mathrm{g})$-interval connected, then the proper value for $\tilde{D}$ is $2d(\mathrm{g}, k)$, that is, twice the smallest integer d such that $\mathrm{g}^{(d)}(1) > k/2$.

The algorithm resulting from combining the k-committee approach with the fast T-token dissemination algorithm is given below. For the sake of conciseness we merge the leader election part into the committee formation part: instead of first electing leaders and then forming committees, all nodes now begin the committee formation phase assuming they are leaders, but stop acting as leaders if at any point they hear a UID smaller than their own. (The original algorithm from Section 3.4.1 can also be written this way; it was presented in two parts only for the sake of clarity.)

At the end of the algorithm each node outputs a committee ID and a set of tokens; the committee IDs always form a correct solution to k-committee, and if $k \geq n$ then the tokens are guaranteed to be a correct solution to all-to-all token dissemination. We follow the algorithm below with the k-verification algorithm (Algorithm 3.2), which allows nodes to determine whether $k \geq n$ and, as a result, whether they have in fact collected all tokens. We assume as usual that the tokens being collected are UIDs; if not, we can run the algorithm below with pairs of the form (UID, input token) instead of UIDs.

Theorem 3.26. *Algorithm 3.6 above solves k-committee in $O(k+k\cdot d(\mathrm{g},k)/T)$ rounds in $(2T,\mathrm{g})$-interval connected graphs. If $k \geq n$, the algorithm also solves all-to-all token dissemination. When combined with the k-verification protocol (Algorithm 3.2), we obtain an $O(n + n \cdot d(\mathrm{g},n)/T)$-round protocol for counting and all-to-all token dissemination.*

Proof. Let us show first that the committee IDs output by the nodes form a correct solution to the k-committee problem. Throughout the algorithm, each node issues at most $T \cdot (\lceil k/T \rceil - 1) + k - T \cdot (\lceil k/T \rceil - 1) = k$ invitations: even if a node remains a leader throughout the execution of the algorithm, it can issue at most T invitations in each of the first $\lceil 2d(\mathrm{g},k)/T \rceil - 1$ cycles, and $k - \lceil k/T \rceil \cdot T$ invitations in the last cycle. As before, nodes that "invite themselves" to join their own committee in the last step of the algorithm are nodes that never extended an invitation to any other node, and in this case the committee formed is a singleton. Thus, no committee contains more than k nodes.

Now suppose that $k \geq n$. In this case we know from Lemma 3.24 that any call to `disseminate` succeeds in spreading the T smallest tokens in the input to all the nodes in the network. In our case, in each cycle of committee formation, in the polling phase we successfully disseminate the UIDs of the T smallest nodes that have not yet joined a committee (or of all remaining nodes if there are fewer than T). These UIDs reach all nodes, and all nodes add them to their *tokens* variable. In particular, exactly one node remains a leader at the end of the first polling phase — the node with the smallest UID. In the invitation phase we successfully disseminate invitations to the T smallest nodes (or to all remaining nodes in the last cycle), and these nodes then join the leader's committee. At the end of the $\lceil k/T \rceil \geq n/T$ cycles, all n nodes have been invited to join the leader's committee, and all have received their invitations and joined. Further, during the polling phases all nodes add the UIDs of all other nodes to their *tokens* set. Therefore all nodes join the same committee and output the set of all UIDs as their token set. □

3.5 Other Variations on the Model

In the sections above we saw that counting and token dissemination are possible under fairly weak assumptions: the graph is 1-interval connected but may change arbitrarily

Algorithm 3.6: k-committee in $(2T, \mathrm{g})$-interval connected graphs

```
1  leader ← true
2  committee ← ⊥
3  tokens ← ∅
   // Committee formation
4  for $i = 1, \ldots, \lceil k/T \rceil - 1$ do
       // Polling phase
5      if committee = ⊥ then
6          candidates ← {self}
7      else
8          candidates ← ∅
9      candidates ← disseminate(candidates, $T$, $2d(\mathrm{g}, k)$)
10     tokens ← tokens ∪ candidates
11     if min(candidates) < self then // Stop acting as a leader
12         leader ← false
       // Leaders issue invitations
13     if leader then
           // Leaders invite the $T$ smallest IDs they collected
           // (or fewer in the final cycle, so that the total does not
              exceed $k$)
14         if $i < \lceil k/T \rceil - 1$ then
15             selected ← smallest-$T$(candidates)
16         else
17             $m \leftarrow k - (\lceil k/T \rceil - 1) \cdot T$
18             selected ← smallest-$m$(candidates)  .
19         invitations ← {self} × selected
20     else
           // Non-leaders do not invite anybody
21         invitations ← ∅
       // Invitation phase
22     invitations ← disseminate(invitations, $T$, $2d(\mathrm{g}, k)$)
       // Invited nodes join committees
23     if $(u_\ell, self)$ ∈ invitations for some $u_\ell$ then
24         committee ← $u_\ell$
25 if committee = ⊥ then
26     committee ← self
27 output (committee, tokens)
```

from round to round, and nodes initially know only their own UID. We now show that these tasks remain solvable (in some cases efficiently) even when the model is further weakened in several ways.

3.5.1 Graphs with Unknown Interval Connectivity and Vertex Growth

Algorithm 3.6 above assumes that all nodes know the degree of interval connectivity present in the communication graph; if the graph is not $2T$-interval connected, invitations may not reach their destination, and the committees formed may contain fewer than k nodes even if $k \geq n$. Similarly, the vertex growth g must be known to all nodes. However, the protocol has "one-sided error": if the dynamic graph is not $(2T, \mathrm{g})$-interval connected, then the committees formed may be too small, but they can never contain more than k nodes, simply because no node ever issues more than k invitations. When we execute the k-verification protocol (Algorithm 3.2) after Algorithm 3.6, we may falsely conclude that $k < n$ even when this is not the case, but we will never conclude that $k \geq n$ when $k < n$. A mistaken verdict of $k < n$ does not lead us to halt with an incorrect answer, it only prevents us from halting with the current value of k; in other words, feeding in wrong values for T or g impacts the *termination* (or *liveness*) of the counting algorithm, not its *safety*. We can use this fact to adapt our counting/token-dissemination algorithm to unknown stability and vertex growth.

Let us focus first on unknown stability and not assume any non-trivial vertex growth: assume that the graph is $2T$-interval connected, but T is unknown. Recall that our overall scheme is to first run the k-committee protocol with some parameters k and T (in this case g is set to the trivial growth function, $\mathrm{g}(s) = s + 1$), and follow with the k-verification protocol, which returns 1 iff $k \geq n$. An output of 0 is interpreted to mean that k is too small and should be increased. If the degree of interval connectivity is unknown, and we run the k-committee protocol with parameters k and T which are both guesses, we must interpret an output of 0 as meaning that *either k is too small, or T is too large.* In this case we either increase k or decrease T, and try again.

Theorem 3.27. *All-to-all token dissemination (with unknown count) can be solved in $O(\min\{n^2, (n + n^2/T) \log n\})$ rounds in T-interval connected graphs, even if T is not known in advance.*

Proof. The basic approach is shown in Algorithm 3.7. Let $\mathcal{A}(k, T)$ denote a call to the combined k-committee + k-verification protocol (i.e., Algorithm 3.6 followed by Algorithm 3.2) with parameters k and T, where k is a guess for the size of the graph, and T is a guess for the degree of interval connectivity. Assume that the output of $\mathcal{A}(k, T)$ is $\bot$ if k-verification failed (i.e., if Algorithm 3.2 outputs 0), and otherwise the output is the set of tokens collected during the execution of the k-committee algorithm (Algorithm 3.6).

Algorithm 3.7: Token dissemination in T-interval connected graphs, T unknown

```
1 for $i = 1, 2, 4, 8, \dots$ do
2     for $k = 1, 2, 4, \dots, i$ do
3         $tokens \leftarrow \mathcal{A}(k, \lfloor k^2/i \rfloor)$
4         if $tokens \neq \bot$ then
5             return $tokens$
6         end
7     end
8 end
```

Each call to $\mathcal{A}(k, \lfloor k^2/i \rfloor)$ requires $O(k + k^2/\lfloor k^2/i \rfloor) = O(i)$ rounds (note that we always have $k \leq i$), and thus the total time complexity of the i-th iteration of the outer loop is $O(i \log i)$.

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

First consider the case where $\max\{N, \lceil N^2/T \rceil\} = N$, and hence $T \geq N$. When we reach the last iteration of the inner loop, where $k = i = N$, we call $\mathcal{A}(N, N)$. This call must succeed (and return a set of tokens rather than $\bot$), and the algorithm terminates.

Next, suppose that $\lceil N^2/T \rceil > N$. Consider the iteration of the inner loop in which $k = N$. In this iteration we call $\mathcal{A}(N, \lfloor N^2/\lceil N^2/T \rceil \rfloor)$. Since $\lfloor N^2/\lceil N^2/T \rceil \rfloor \leq T$, this again must succeed, and the algorithm terminates.

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

The asymptotic time complexity of Algorithm 3.7 improves upon the original $O(n^2)$ algorithm (which assumes only 1-interval connectivity) only when $T = \omega(\log n)$. However, it is possible to execute both algorithms in parallel, either by doubling the message sizes or by interleaving the steps (e.g., executing one step of Algorithm 3.3 in even rounds and one step of Algorithm 3.7 in odd rounds, and keeping separate copies of the local variables for each). When the first of the two algorithms finishes, the combined algorithm terminates and returns the same answer. This leads to a time complexity of $O(\min\{n^2, (n + n^2/T) \log n\})$. $\square$

We can follow the same approach to adapt to unknown vertex growth. For example, say that we believe the network is likely to be a vertex expander, but we are not certain that it will be and we do not know the expansion constant if it is. Fix some family $H \subseteq \mathbb{R}$ of expansion constants. For each combination of a guess $k = 1, 2, 4, 8, \dots$ and an expansion constant $\alpha \in H \cup \{0\}$, we can execute the k-committee + k-verification protocol under the assumption that the graph is $(1, \mathrm{g}_\alpha)$-interval connected, where $\mathrm{g}_\alpha(s) := \alpha \cdot s$ if $\alpha \neq 0$ and $\mathrm{g}_\alpha(s) := 1$ if $\alpha = 0$ (that is, $\alpha = 0$ represents the case where the graph is not an expander). If k-verification succeeds we terminate, and otherwise we either increase k or decrease α, and try again. We pay an additional $\log|H|$ factor in the running time, so if the graph is not an expander, the total round complexity will be $O(n^2 \log|H|)$. However, if the graph *is* a vertex expander with expansion constant at least α for some $\alpha \in H$, then the total round complexity is $O(n \log_\alpha(n) \log|H|)$. Along similar lines one might try to guess various combinations of stability parameter T and vertex growth g such that the graph is (T, g)-interval connected; we always pay a log factor in the worst case, but the potential gain when the network is well-behaved can be large.

3.5.2 Coping with Asynchronous Wakeup

A common strategy for conserving power during period of inactivity in a wireless network is to have nodes switch their hardware radios to a low-powered "sleeping" state, where instead of sending and receiving messages nodes sense the channel and only "wake up" upon sensing activity. We model this behavior as *asynchronous start* (see Section 2.1): instead of assuming that all nodes begin the computation at the same time, as we have done until now, in asynchronous wakeup, computation is initiated by one or more nodes, and the remaining nodes are initially asleep. A node wakes up when it receives a message from some node that is already awake, and then it joins the computation. However, since it is already "too late" to send messages in the current round, the awakened node sends its first message only in the next round.

It is not hard to see that under asynchronous wakeup, no meaningful global computation can be accomplished. We use consensus as the essential example for "meaningful global computation", since it is a fairly weak task: it only requires nodes to agree on a single bit, and the output is only constrained under two specific input assignments (the all-zero and the all-one inputs).

Proposition 3.28. *If there is no* a priori *upper bound on the count, consensus cannot be solved under asynchronous wakeup: there is no randomized algorithm that achieves both validity and agreement with probability* $\geq 3/4$ *in every execution.*

Proof. We use the classical *partitioning argument*: we construct an execution where two parts of the network cannot observe each other's existence, and show that agreement is violated, as the nodes in one part must decide 0 and the nodes in the other must decide 1.

Suppose for the sake of contradiction that $\mathcal{A}$ is a randomized algorithm that solves consensus under asynchronous wakeup, and let s be the worst-case number of rounds required for $\mathcal{A}$ to halt with probability 1/2 in networks of size 1. Now consider an undirected network $G = (V, E)$ over $s + 2$ nodes $V = \{1, \ldots, s+2\}$, where initially only nodes 1 and 2 are awake, the input to node 1 is 0 and the input to node 2 is 1. The edges are defined as follows: at each time t we have a pool of $s - t$ sleeping nodes (initially we have s, since only 1 and 2 are awake); in round $t + 1$ we choose some sleeping node u, and we connect all the active nodes only to u. In addition, the sleeping nodes are connected amongst themselves in a line. Since all active nodes are connected only to u, only node u wakes up in round $t + 1$; however, it does not get to send a message until the next round, and by that point it is no longer connected to any active node. Therefore node 1 cannot distinguish this execution from an execution α where it is the only node in the network, and node 2 also cannot distinguish this execution from an execution β where *it* is the only node in the network. In executions α and β, validity requires nodes 1 and 2 to output 0 and 1 respectively, and by choice of s, each of the two nodes halts by time s with probability at least $1/2$ (independently of the other node). Thus, after at most s rounds in G, with probability at least $1 - (1/2)^2 = 3/4$, nodes 1 and 2 halt and output 0 and 1 respectively, violating agreement. □

Corollary 3.29. *Counting (exact or approximate) and all-to-all token dissemination cannot be solved under asynchronous wakeup in the absence of an* a priori *bound on the size of the network.*

Proof. We saw in Section 3.1 that counting reduces to all-to-all token dissemination, and computing a minimum reduces to counting. Consensus is easily reduced to minimum by having all nodes decide on the minimum input. Thus the impossibility of the "harder" tasks follows from the impossibility of consensus. □

The impossibility of consensus is an artifact of our round model: since nodes wake up after receiving a message, they cannot provide any feedback to the node(s) that woke them up, since at that point it is too late in the round to *send* messages. Thus, the node initiating the computation can labor under the illusion that it is all by itself in the network, while in fact there are many other nodes that wake up but are immediately whisked away by the adversary. We could eliminate this scenario in several ways: e.g., we could change the model to allow nodes that wake up to respond immediately, or allow the graph to change only once every two rounds. This would allow every node that wakes up to alert the node that awakened it. However, it turns out that a weaker guarantee is sufficient: we only need *some* node that wakes up to alert the node that awakened it, and for this purpose 2-interval connectivity suffices, provided the graph is undirected.

Theorem 3.30. *In 2-interval connected undirected graphs with asynchronous wakeup, all-to-all token dissemination with unknown count can be solved in $O(n^2)$ rounds.*

Proof. We modify the definition of k-committee and the algorithms from Sections 3.3–3.4 as follows.

k-committee with wakeup: Each awake node outputs a committee ID, such that no committee contains more than k nodes. Sleeping nodes are not counted as belonging to any committee. If $k \geq n$, we require all nodes to belong to the same committee, and in particular all nodes must be awake at the end of the protocol.

Timestamps: Since each node wakes up at a different time, we no longer have a global time-reference that all nodes are aware of; instead, each node knows only its local time, i.e., how many rounds have passed since it woke up. To resolve this issue, we attach to each message a *timestamp* containing the local time at the sender. Note that each part of the combined k-committee/k-verification protocol requires a fixed number of rounds that depends only on k. Therefore the timestamp is sufficient to know which step of the combined k-committee/k-verification protocol the sender just executed, including the value of k, the current line number in Algorithm 3.3, etc. Nodes always strive to be up-to-date: when they receive a message with a timestamp larger than their local time, they update their local time to match the timestamp, and jump to the step that corresponds to the new local time. In particular, a node that wakes up jumps to the step indicated by the timestamp on the message that awakened it (or the largest timestamp if it received more than one message). Jumping into the middle of a phase requires several modifications to the algorithm, which we describe below.

The k-committee algorithm: Algorithm 3.3 is modified as follows.

- All UIDs stored and sent are replaced with pairs (timestamp, UID); where nodes use their own UID, they use their local clock as the timestamp.
- Wherever nodes compare UIDs (to select leaders, etc.), they now lexicographically compare the pair (timestamp, UID), giving precedence to nodes with a *larger* local time (i.e., nodes that woke up earlier).
- A node that jumps into the leader election part of Algorithm 3.3 participates normally: it sets *min_uid* to the value it received in the message that woke it up and follows the remainder of the protocol.
- A node that jumps to any step of Algorithm 3.3 following the leader election "sits out" the current value of k: instead of actively participating in the protocol, it only forwards messages it receives (choosing an arbitrary message if there is more than one), and at the end it forms its own committee, outputting its own UID as its committee ID. From this point on it participates normally in the remainder of the execution (unless it later makes another jump).

The modified protocol solves our modified version of k-committee: it is still true that no committee contains k members, because no node issues more than k invitations. And if $k \geq n$, then the leader election part takes up at least $n - 1$ rounds, which is sufficient to wake up all the nodes in the graph and have them set their clocks to the largest clock in the network. Since nodes give precedenc toe nodes with a larger local time when they compare (UID,timestamp) pairs, at the end of the leader election part, only the smallest node among the nodes that initiated the computation survives as a leader; this node successfully invites all the other nodes to join its committee.

k-verification: We follow the original k-verification protocol, except that we execute it for $2k$ rounds instead of k rounds. Nodes that jump into the middle of the protocol set their *cid* to $\bot$ and participate normally (i.e., they send $\bot$ in all subsequent rounds).

If we start from an initial state where the *cid*s represent a legal solution to k-committee, then k-verification succeeds: if $k \geq n$, then initially all nodes are awake and have the same cid, so as in the synchronous-wakeup case, all nodes output 1. If $k < n$, we can show that every two rounds, each committee shrinks by at least one: from 2-interval connectivity, each committee has some edge $\{u, v\}$ that persists for two rounds, such that u belongs to the committee and v either belongs to a different committee, or has $cid_v = \bot$, or is asleep. In the first two cases we have $cid_v \neq cid_u$, so v sends a value that causes u node to drop out of the committee in the first round. In the second case, v wakes up in the first round, and sends $\bot$ in the second round, causing u to drop out of the committee. Since each committee initially contains at most k nodes, after $2k$ such rounds all committees are empty, and all nodes output 0. □

3.5.3 Counting and Token Dissemination with Beeps

As a curiousity, we observe that the problem of exact counting can be solved (albeit very inefficiently) in 1-interval connected graphs even if nodes have rudimentary radios that are only capable of making and detecting noise on the channel. This model is called "the beep model" [37, 2]. More precisely, in each round, nodes can decide whether to "beep" (broadcast some signal or cause noise) or not, and their feedback for the round is "noise" if they or some node in their in-neighborhood beeped in the round, and otherwise it is "silence".

To solve counting in the beep model, we proceed as follows: we devote a phase comprising k rounds to each "guess" $k = 1, 2, 4, 8, \ldots$ for the count. During these k rounds, each node whose UID is greater than k beeps in every round, and every node that hears noise beeps in all subsequent rounds of the phase. At the end of the phase, if the node heard noise at any point, it moves on to the next value of k; otherwise it concludes that $k \geq n$. At this point we can narrow in on the exact count by using binary search. Correctness follows from the simple fact that since UIDs are unique, there can be at most k nodes that have UIDs no greater than k; if $k < n$, then k rounds suffice for all of these nodes to hear noise, just as we saw in the k-verification protocol. We note that this procedure is extremely inefficient, as the largest UID in the graph may be much larger than the number of nodes in the graph.

Once the count is determined, nodes are able to compute the minimum over inputs encoded in ℓ bits in $n \cdot \ell$ rounds: initially all nodes compete to have their input selected; for each bit $i = 1, \ldots, \ell$, remaining competitors whose input has 0 in position i beep for n rounds, while all other nodes simply forward beeps they hear (i.e., they remain silent until they hear noise, then they start beeping). After n rounds, the i-th bit of the minimum input is determined to be 0 if noise was heard at any point, and 1 otherwise. If the i-th bit is determined to be 0, any competitor whose input has 1 in the i-th position stops competing. In this manner all nodes can learn the value of the smallest input in $n \cdot \ell$ rounds. To solve all-to-all token dissemination, nodes can repeat this process n times, disseminating first the smallest token, then the next-smallest token, and so on. Nodes whose token was already disseminated (i.e., was selected as the minimum) in some iteration do not compete in subsequent iterations, to ensure that we disseminate a new token in each iteration.

Theorem 3.31. *Counting and token dissemination can be solved in $O(N \log N + n^2 \log n)$ rounds in the beep model, where N is the largest UID present in the execution.*

3.6 Randomized Approximate Counting

Until now we considered deterministic and exact counting. We now show that against an *oblivious* adversary, randomization allows us to compute an approximate count in linear time. We assume that the nodes know some potentially loose upper bound N on n. Recall from Chapter 2 that an oblivious adversary commits to the entire dynamic graph in advance, before the nodes make any random choices.

High-level overview of the algorithm. Our algorithm is based on a technique described in [114] in the context of gossiping protocols.[3] The main idea is to use the fact that the minimum of n iid exponential random variables with rate 1 is itself an exponential variable with rate n. If we have each node draw an exponential random variable with rate 1, then the expected value of the minimum is $1/n$, and we can use this to approximate the count. To amplify the probability of obtaining an accurate estimate we repeat this process several times in parallel, and compute our estimate from the average result. Formally, given a set S of ℓ-tuples of independent exponential variables with rate 1, $S = \left\{ \left(Y_1^1, \ldots, Y_1^\ell\right), \ldots, \left(Y_{|S|}^1, \ldots, Y_{|S|}^\ell\right) \right\}$, the estimate we compute from S is

$$\hat{n}(S) := \frac{\ell}{\sum_{i=1}^{\ell} \min_{1 \le j \le |S|} Y_j^i}. \tag{3.6.1}$$

In [114], the approach above was used to approximate the sum of a known number of inputs. For our purposes, their technique suggests the following scheme: to estimate $|V|$, each node v initially computes an ℓ-tuple of exponential random variables with rate 1, $(Y_v^1, \ldots, Y_v^\ell)$. The objective of the nodes is to compute $\hat{n}(V)$. (Here and in the sequel, we abuse notation slightly be using $\hat{n}(U)$, where $U \subseteq V$, to denote the estimate $\hat{n}(S_U)$ computed from the set $S_U = \left\{ (Y_v^1, \ldots, Y_v^\ell) \mid v \in U \right\}$.) To compute the estimate, each node u computes the pointwise minimum

$$(\min_{v \in S} Y_v^1, \min_{v \in S} Y_v^2, \ldots, \min_{v \in S} Y_v^\ell),$$

where S is the set of nodes whose values u has heard, and then node u computes $\hat{n}(S) = \ell / \sum_{i=1}^{\ell} \min_{v \in S} Y_v^i$. The pointwise minimum is easily computed by simply forwarding the smallest value heard for each coordinate.

How should we decide when to terminate?[4] At time t, the only information avail-

[3]A gossip protocol operates in a complete graph, with each node choosing a random neighbor to contact in each round. Such protocols, including the one in [114], typically use messages of unbounded size.

[4]This is a challenge that does not arise in [114], where nodes never halt and the sum is computed in the limit.

able to node u is the estimate $\hat{n}(S)$ it has computed so far, where $S = \mathsf{past}(u,t)_0$ is the set of nodes from which u has heard. Since we want to estimate the size of the entire network, we wish node u to terminate only when $\mathsf{past}(u,t)_0 = V$. Recall that in Section 3.2, we used the following termination test:

$$|\mathsf{past}(u,t)_0| \overset{?}{\leq} t \tag{3.6.2}$$

as a condition which, when true, indicates that $\mathsf{past}(u,t)_0 = V$ and we are allowed to halt. We will now use the same idea: to check at time t whether $\mathsf{past}(u,t)_0 = V$, node u examines its estimate $\hat{n}_u$ and compares it to the time t, taking into account the inaccuracy of $\hat{n}_u$, which is bounded with high probability by ε. If $\hat{n}_u < (1-\varepsilon)t$, node u concludes that with high probability condition (3.6.2) above is true, and therefore $\mathsf{past}(u,t)_0 = V$; in this case node u halts and outputs its current estimate. Otherwise node u continues until the termination test succeeds.

The technical details. In [114], an easy application of Cramér's Theorem yields the following bound on the error of the average-based estimate:

Lemma 3.32 ([114]). *Let S be a set of ℓ-tuples of independent exponential variables with rate 1, $S = \left\{(Y_1^1, \ldots, Y_1^\ell), \ldots, \left(Y_{|S|}^1, \ldots, Y_{|S|}^\ell\right)\right\}$. Define*

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

Then

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

In our case the number of repetitions ℓ is chosen as follows: given parameters $\epsilon \in (0, 1/2)$ and $c \geq 1$ representing the approximation error and the constant in the success probability $1 - 1/N^c$, respectively, we define

$$\ell := \lceil (2+2c) \cdot 27 \ln(N)/\varepsilon^2 \rceil. \tag{3.6.3}$$

Plugging this value into Lemma 3.32, we obtain

$$\begin{aligned} \Pr\left[\left|\hat{n}(S) - |S|\right| > \frac{2}{3}\varepsilon \cdot |S|\right] &\leq 2e^{-\varepsilon^2 \ell/27} \leq 2e^{-3-(2+c)\ln N} \\ &< \frac{1}{4N^{2+c}}. \end{aligned} \tag{3.6.4}$$

Let us now describe two technical issues having to do with the size of messages sent by the algorithm. First, sending exact values for the variables $Y_i^{(v)}$ would require nodes to send real numbers, which cannot be represented using a finite number of bits. Therefore we round and truncate the variables Y_v^i as follows:

$$\tilde{Y}_v^i := \min\left\{\frac{1}{4\ell N^{1+c}}, \max\left\{\ln(4\ell N^{1+c}), \left(1+\frac{\varepsilon}{4}\right)^{\lfloor \log_{1+\varepsilon/4}(Y_v^i)\rfloor}\right\}\right\}. \tag{3.6.5}$$

In other words, we round Y_v^i down to the next smaller integer power of $1+\varepsilon/4$, and restrict it to the range $[1/(4\ell N^{1+c}), \ln(4\ell N^{1+c})]$. We will show that with high probability, all variables Y_v^i are already in this range, and thus restricting the range only has an effect with negligible probability. Since $\tilde{Y}_v^i$ is an integer power of $1+\varepsilon/4$ in the range $[1/(4\ell N^{1+c}), \ln(4\ell N^{1+c})]$, it can be represented using $O(\log\log_{1+\varepsilon/4}(\ell N)) = O(\log\log N + \log(1/\varepsilon))$ bits.

In order to further reduce message sizes, we may wish to avoid sending a complete tuple $(Y_v^1, \ldots, Y_v^\ell)$ in a single message; we can trade off time against message size by sending the tuple over several rounds. To do this we partition the variables $Y_v^1, \ldots, Y_v^\ell$ into subsets of b variables each, where b is a parameter of the algorithm (and $1 \leq b \leq \ell$). We assume for simplicity that b is an integer divisor of ℓ (if not, ℓ can be increased to the nearest multiple of b without reducing the success probability), and let $m := \ell/b$ be the number of subsets. Now, instead of sending the entire tuple $(\min_{v\in S} Y_v^1, \ldots, \min_{v\in S} Y_v^\ell)$, we iterate through the subsets, sending only coordinates $i\cdot b+1, i\cdot b+2, \ldots, i\cdot b+b$ (that is, coordinates corresponding to the $(i+1)$-st subset) in each round r such that $(r-1) \bmod m = i$. The total runtime of the algorithm is increased by a factor of b. A slight complication arises from the fact that now different coordinates are sent in different rounds, which means they are broadcast over different communication graphs. However, we will see that this issue is easily dealt with.

The final algorithm is given in Algorithm 3.8. We use $\underline{X}$ to denote a tuple, and X_i to denote the i-th coordinate of the tuple. As usual, we omit the subscript v we give the code executed by node v.

Note that each iteration of the outer loop ($r = 1, 2, \ldots$) spans m rounds. Thus, at time $m\cdot r$ the nodes complete the r-th iteration of the outer loop and compute a new estimate $\tilde{n}(r)$. We are not interested in intermediate values encountered during the execution of the inner loop. To simplify our notation we use $\tilde{n}_u(r)$ to denote $\tilde{n}_u(m\cdot r)$, that is, the estimate computed by u at the end of the r-th iteration of the outer loop.

Because we partition the ℓ coordinates into subsets and send only one subset in each round, different coordinates j reflect the initial values Y_v^j of different sets of nodes v. To capture the set of nodes that influence a specific coordinate, we define a *restriction* of the dynamic graph: given a graph $G = (V, E)$, for each $i = 1, \ldots, m$ we define $G|_i := (V, E|_i)$, where $E|_i(r) = E(m\cdot(r-1)+i)$. In other words, round r in $G|_i$ corresponds to the round in which subset i is sent for the r-th time in the original execution in G. Now let

$$C_u^i(t) := \mathsf{past}(u, 1+\lfloor (t-i)/m\rfloor)_{G|_i,0} = \left\{v \in V \mid (v,0) \rightsquigarrow_{G|_i} (u, 1+\lfloor (t-i)/m\rfloor)\right\}.$$

An easy induction on t shows that for any node $u \in V$, subset $i = 1, \ldots, m$, index

Algorithm 3.8: Randomized approximate counting in linear time, code for node v

```
$\underline{Z} \leftarrow (\tilde{Y}^1, \ldots, \tilde{Y}^\ell)$
for $r = 1, 2, \ldots$ do
    for $i = 1, \ldots, m$ do
        // Send the $i$-th subset
        $\underline{W} \leftarrow (Z_{(i-1)b+1}, \ldots, Z_{(i-1)b+b})$
        broadcast $\underline{W}$
        receive $\underline{W_{v_1}}, \ldots, \underline{W_{v_s}}$ from neighbors
        // Update each coordinate in subset $i$ to the minimum value
        heard for that coordinate
        for $j = (i-1)b + 1, \ldots, (i-1)b + b$ do
            $Z^j \leftarrow \min\{Z^j, W^j_{v_1}, \ldots, W^j_{v_s}\}$
        end
    end
    $\tilde{n}(r) \leftarrow \ell / \sum_{i=1}^{\ell} Z^i$
    if $(1-\varepsilon)r > \tilde{n}(r)$ then terminate and output $\tilde{n}(r)$
end
```

$j = 1, \ldots, b$ inside subset i, and time t, we have

$$Z_u^{(i-1)b+j}(t) = \min_{v \in C_u^i(t)} \tilde{Y}_v^{(i-1)b+j}.$$

Now we are ready to prove the correctness of the algorithm. Recall that each node v initially computes ℓ exponential random variables $Y_v^1, \ldots, Y_v^\ell$, and then truncates and rounds them according to (3.6.5). We first analyze the algorithm ignoring truncation, and then show that w.h.p. truncation does not change the output. Let

$$\dot{Y}_v^j := \left(1 + \frac{\varepsilon}{4}\right)^{\lfloor \log_{1+\varepsilon/4}(Y_v^j) \rfloor} \tag{3.6.6}$$

denote the value of Y_v^j after rounding but before truncation. Note that

$$\dot{Y}_j^{(v)} \le Y_j^{(v)} \le \left(1 + \frac{\varepsilon}{4}\right) \cdot \dot{Y}_j^{(v)}. \tag{3.6.7}$$

Consider an algorithm $\mathcal{A}'$ which is identical to Algorithm 3.8 except that nodes initialize $\underline{Z_v}$ using $(\dot{Y}_v^1, \ldots, \dot{Y}_v^\ell)$ instead of $(\tilde{Y}_v^1, \ldots, \tilde{Y}_v^\ell)$. Let $\dot{n}_v(r)$ denote the estimate computed by node v at time $r \cdot m$ in $\mathcal{A}'$ (as opposed to $\tilde{n}_v(r)$, the actual estimate computed by node v in Algorithm 3.8). We first show that if the algorithm terminates at time t, then $\dot{n}_v(t)$ is a good approximation for n w.h.p., and then we show that w.h.p. we have $\dot{n}_v(t) = \tilde{n}_v(t)$ for all nodes v.

The first step in the analysis is to bound the nodes' estimates from below, and show that nodes are not likely to halt before time n: with high probability, at all times preceding time n, each node's estimate is large enough to prevent it from halting.

Lemma 3.33. *For each node $v \in V$ and iteration $r \leq n$, with probability at least $1 - \frac{1}{4N^{2+c}}$ we have*

$$\dot{n}_v(r) \geq (1-\varepsilon)r.$$

Proof. Recall that for each specific coordinate j in subset $i = \lceil j/m \rceil$, we have $Z_v^j = \min_{u \in C_v^i(t)} \tilde{Y}_u^j$, so intuitively the quality of the estimate $\dot{n}_v(r)$ is closely related to the size of the sets $C_v^i(t)$ for different i. These sets cannot be too small; from Lemma 2.21, for any subset i we have

$$|C_v^i(t)| \geq \min\left\{n, 1 + \lfloor\frac{t-i}{m}\rfloor\right\} = \min\left\{n, 1 + \lfloor\frac{r \cdot m - i}{m}\rfloor\right\} = \min\{n, r\}. \quad (3.6.8)$$

For each $i = 1, \ldots, m$, we choose r nodes $v_1^i, \ldots, v_r^i \in C_v^i(t)$ (if $|C_v^i(t)| > r$ the choice is arbitrarily). Let $S = \{\underline{Y_1}, \ldots, \underline{Y_r}\}$, where for each $k = 1, \ldots, r$,

$$\underline{Y_k} = \left(Y_{v_k^1}^1, \ldots, Y_{v_k^1}^b, Y_{v_k^2}^{b+1}, \ldots, Y_{v_k^2}^{2b}, \ldots, Y_{v_k^m}^{(m-1)b+1}, \ldots, Y_{v_k^m}^{mb}\right).$$

Due to rounding and to the fact that we might have $|C_v^i(t)| > r$, the estimate $\dot{n}_v(r)$ computed by node v at time t is not quite $\hat{n}(S)$, but it is not too far off. For any coordinate $j = 1, \ldots, \ell$ that belongs to subset $i = \lceil j/m \rceil$,

$$Z_v^j = \min_{u \in C_v^i(t)} \dot{Y}_u^j \leq \min_{k=1,\ldots,r} \dot{Y}_{v_k^i}^j \overset{(3.6.7)}{\leq} \min_{k=1,\ldots,r} Y_{v_k^i}^j.$$

It follows that

$$\dot{n}_v(r) = \frac{\ell}{\sum_{j=1}^{\ell} Z_j^{(u)}} \geq \frac{\ell}{\sum_{j=1}^{\ell} \min_{k=1,\ldots,r} Y_j^{(v_k^i)}} = \hat{n}(S).$$

Consequently we obtain

$$\begin{aligned}\Pr\left[\dot{n}_v(r) < (1-\varepsilon)r\right] &\leq \Pr\left[\hat{n}(S) < (1-\varepsilon)r\right] \\ &\leq \Pr\left[\left|\hat{n}(S) - r\right| > \varepsilon \cdot r\right] \overset{(3.6.4)}{\leq} \frac{1}{4N^{2+c}}. \end{aligned} \quad (3.6.9)$$

□

Lemma 3.33 shows that w.h.p., nodes do not halt prematurely, and when they are ready to halt (at time n), their estimate is not too small compared to n. We now bound the estimate from above and show that starting from time n, the estimate is w.h.p. not too large compared to n.

Lemma 3.34. *For each node $v \in V$ and iteration $r \geq n$, with probability at least $1 - \frac{1}{4N^{2+c}}$ we have*

$$\dot{n}_v(r) = \dot{n}_v(n) \leq (1+\varepsilon)n.$$

Proof. The proof is quite similar to Lemma 3.33. In fact the two proofs can be combined, since (3.6.4) gives us both an upper and a lower bound on the estimate. However, at times $m \cdot r < n$ we do not obtain a meaningful upper bound as a function of r, as the sets $C_v^i(r)$ of nodes whose values "go into the estimate" may contain more than r nodes. For clarity, we separate the lower bound (Lemma 3.33), which bounds the estimate in terms of r, from the upper bound (the current lemma), which applies only starting from time n and bounds the estimate in comparison to n.

Let $S = \left\{\left(Y_v^1, \ldots, Y_v^\ell\right) \mid v \in V\right\}$ be the set of all ℓ-tuples initially computed by the nodes, and let $\dot{S} = \left\{\left(\dot{Y}_v^1, \ldots, \dot{Y}_v^\ell\right) \mid v \in V\right\}$ be the set of ℓ-tuples of rounded values. If t is a time that falls in iteration $r \geq n$, then from (3.6.8) we know that for all i we have $|C_v^i(t)| = n$, and hence for any coordinate j in subset i,

$$Z_v^j = \min_{u \in C_v^i(t)} \dot{Y}_v^j = \min_{u \in V} \tilde{Y}_u^j \overset{(3.6.7)}{\geq} \frac{\min_{u \in V} Y_u^j}{1 + \frac{\varepsilon}{4}}.$$

In particular, the estimate $\dot{n}_v(r)$ does not change after time $t = n \cdot m$ (that is, after the loop counter r reaches n). Thus, at any time $t \geq n \cdot m$ we have

$$\dot{n}_v(r) = \dot{n}_v(n) = \hat{n}(\dot{S}) = \frac{\ell}{\sum_{j=1}^{\ell} Z_j^{(u)}} \leq \left(1 + \frac{\varepsilon}{4}\right) \frac{\ell}{\sum_{j=1}^{\ell} \min_{v \in V} Y_j^{(v)}} = \left(1 + \frac{\varepsilon}{4}\right) \hat{n}(S).$$

Now let us bound the probability that $\dot{n}_v(r)$ is too large:

$$\begin{aligned}
\Pr\left[\dot{n}_v(r) > (1+\varepsilon)n\right] &\leq \Pr\left[\left(1 + \frac{\varepsilon}{4}\right)\hat{n}(S) > (1+\varepsilon)n\right] \\
&= \Pr\left[\left(1 + \frac{\varepsilon}{4}\right)(\hat{n}(S) - n) > \frac{3}{4}\varepsilon n\right] \\
&\overset{(\varepsilon<1/2)}{\leq} \Pr\left[\left(1 + \frac{\varepsilon}{4}\right)(\hat{n}(S) - n) > \left(1 + \frac{\varepsilon}{4}\right) \cdot \frac{2}{3}\varepsilon n\right] \\
&\leq \Pr\left[\left|\hat{n}(S) - n\right| > \frac{2}{3}\varepsilon n\right] \\
&\overset{(3.6.4)}{\leq} \frac{1}{4N^{2+c}}.
\end{aligned}$$

□

Theorem 3.35. *For $\varepsilon \in (0, 1/2)$ and $c > 0$, with probability at least $1 - 1/N^c$, every node of Algorithm 3.8 outputs the same value $\tilde{n}$, and this value satisfies $|\tilde{n} - n| \leq \varepsilon n$.*

Proof. Let $B_{v,r}$ be the event that at node v, if $r \leq n$ then we have $\dot{n}_v(r) \geq (1-\varepsilon)r$, and if $r \geq n$ then we have $\dot{n}_v(r) \leq (1+\varepsilon)n$. Lemmas 3.33 and 3.34 together show that

$$\Pr\left[B_{v,r}\right] \geq 1 - \left(\frac{1}{4N^{2+c}} + \frac{1}{4N^{2+c}}\right) = 1 - \frac{1}{2N^{2+c}}.$$

Also, let A be the event that for all $v \in V$ and $j = 1, \ldots, \ell$ we have $\tilde{Y}_v^j = \dot{Y}_v^j$; that is, all the exponential variables are within the range $[1/(4\ell N^{1+c}), \ln(4\ell N^{1+c})]$, so truncating them has no effect. For a single exponential variable X with rate 1 we have

$$\Pr\left[\dot{X} \neq \tilde{X}\right] = e^{-\ln(4\ell N^{1+c})} + 1 - e^{-1/(4\ell N^{1+c})}$$
$$\leq \frac{1}{4\ell N^{1+c}} + \frac{1}{4\ell N^{1+c}} = \frac{1}{2\ell N^{1+c}}.$$

Since there is a total of $n \cdot \ell$ exponential variables in the system, a union bound yields

$$\Pr[A] \geq 1 - n \cdot \ell \cdot \frac{1}{2\ell N^{1+c}} \geq 1 - \frac{1}{2N^c}.$$

If A occurs then for all $v \in V$ and $r \in \mathbb{N}^+$ we have $\tilde{n}_v(r) = \dot{n}_v(r)$. If, in addition to A, all events $B_{v,r}$ for $v \in V$ and $r = 1, \ldots, n$ occur, then

(a) No node v halts and outputs $\tilde{n}_v(r)$ if $r < n$ (because in that case $\tilde{n}_v(r) = \dot{n}_v(r) \geq (1-\varepsilon)r$ at all nodes),

(b) All nodes halt at time $(((1+\varepsilon)/(1-\varepsilon))n+1)m$ (because $\tilde{n}_v(n) = \dot{n}_v(n) \leq (1+\varepsilon)n$, and $\tilde{n}$ does not change once r reaches n), and

(c) All nodes output the same estimate $\tilde{n}_v(n) = \dot{n}_v(n) = \hat{n}(\tilde{S})$, which satisfies $\left|\hat{n}(\tilde{S}) - n\right| \leq \varepsilon n$. Here $\tilde{S}$ is the set of rounded and truncated tuples initially generated by all nodes.

In other words, if A and $\bigcap_{v \in V} \bigcap_{r=1}^{n} B_{v,r}$ occur, then the algorithm terminates in linear time (assuming ε is a constant) and computes an ε-approximation for the count. A union bound shows that the probability that this does *not* occur is bounded by

$$\Pr\left[\overline{A} \cup \bigcup_{v \in V} \bigcup_{r=1}^{n} \overline{B_{v,r}}\right] \leq \frac{1}{2N^c} + N^2 \cdot \frac{1}{2N^{2+c}} = \frac{1}{N^c}.$$

This completes the proof. □

Remark 3.36. As the proof above shows, Algorithm 3.8 has the useful property that with high probability, all nodes halt at the same time and output the same estimate. This allows us to treat ε-approximate counting as a simultaneous task which, once completed, provides all nodes with a consistent bound on the size that all agree on.

Remark 3.37. Algorithm 3.8 is designed to estimate the number of nodes in the network. However, suppose that instead we wish to estimate the number of nodes matching some criteria (e.g., nodes that received a specific value in their input). We can accomplish this by running two instances of Algorithm 3.8 side-by-side. In the first instance, $\mathcal{A}$, all nodes participate. In the second instance, $\mathcal{A}'$, only nodes matching the criteria participate, and the remaining nodes only forward the values they receive. When $\mathcal{A}$ halts, nodes halt and output the estimate computed by $\mathcal{A}'$. We saw in Theorem 3.35 that w.h.p. $\mathcal{A}$ does not halt before time n; it is easy to modify the proof of Theorem 3.35 to show that by this time $\mathcal{A}'$ has obtained an ε-approximation to the number of nodes matching the criterion.

This modification will be useful when we later discuss the relationship between counting and sensitive tasks.

3.7 Lower Bound on Token Dissemination by Token-Forwarding

In this section we focus on the problem of token dissemination with a *known* number of tokens k, and discuss lower bounds on this problem. (The problem is of course no harder than token dissemination with an *unknown* number of tokens.)

The token-dissemination algorithms we presented in this chapter used no "combining": they could store and forward tokens, but not alter or combine them in any way (unlike, e.g., network coding algorithms). We call the class of algorithms that satisfy this property *token-forwarding algorithms.*

Definition 3.38. *A token-dissemination algorithm is called a* token-forwarding algorithm *if it satisfies the following conditions.*

(a) The messages of the algorithm are pairs of the form (x, y), where x is either a token or the special symbol $\perp$ (indicating "no token"), and y contains control information of the algorithm (in arbitrary format).

(b) A node u can send a message (x, y) only if x was in node u's input or node u has previously received a message of the form (x, z). (The control information y is unconstrained.)

(c) Node u can halt only when it has received all tokens in the global input, either in its own input or in messages from other nodes.

Note that this definition allows nodes to interact with their tokens arbitrarily and to compute unrestricted control information. However, informally speaking, control information can only be used *to decide which token to send*; it cannot replace the actual sending of whole, unaltered tokens. We say that a token-forwarding algorithm *uses no control information* if its messages are of the form (x, ε), that is, the control information is always empty.

When this work was originally presented (in [94]), we gave two lower bounds on token-forwarding algorithms:

1. An $\Omega(n \log k)$-round lower bound on centralized token-forwarding algorithms in 1-interval connected graphs.

2. An $\Omega(n + nk/T)$-round lower bound on a restricted class of distributed randomized token-forwarding algorithms (formalized in Definition 3.39). This class of algorithms includes all the algorithms in this chapter.

The first lower bound has since been strengthened in [43] to an $\Omega(nk)$-round lower bound on centralized token dissemination in 1-interval connected graphs, later generalized in [68] to an $\Omega(n + n^2/T^2)$-round bound in T-interval connected graphs. Therefore we do not include our first lower bound here. However, the result of [68] does not subsume our second lower bound, as it is off by a factor of T from our upper bound in Section 3.4.3. We therefore include the second lower bound here.

The lower bound we describe here is against a restricted class of randomized token-forwarding algorithms, which we call *limited-history algorithms*. Informally, in a limited-history algorithm, once a node receives all the tokens, its behavior in all future rounds is fixed and depends only on the history up to that point.

We assume for simplicity that the nodes do not halt, and we are interested in the time required for each node to collect all tokens (this only strengthens our lower bound: in the original problem nodes *are* required to halt, but cannot do so before collecting all tokens). We therefore treat each node's output as an infinite sequence $x_1, x_2, \ldots, \in (X_\perp)^\omega$, where X is the set of tokens and $X_\perp$ denotes the set $X \cup \{\perp\}$ (recall that $\perp$ a special symbol indicating that the node does not send a token).

Definition 3.39 (Limited-history algorithms). *A token-forwarding algorithm is said to be* limited-history *if it satisfies the following condition: let α be an execution prefix comprising $t \geq 0$ rounds, and let u be a node that receives all tokens by time t in α. Then the distribution of tokens sent by u in the rounds following t is the same in all possible extensions of α. That is, there is a fixed mapping*[5]

$$\mathcal{D} : (X_\perp)^* \to [0, 1]$$

such that in any extension of α (regardless of the dynamic graph and the messages received by u), node u outputs each sequence of tokens[6] *$x_1, \ldots, x_m$ in rounds $t + 1, \ldots, t + m$ with probability $\mathcal{D}(x_1, \ldots, x_m)$.*

The class of limited-history algorithms includes all the token-dissemination algorithms in this chapter, as well as the following natural approaches:

- Sending a random token from the set of tokens known in each round,
- Round-robin over tokens (possibly depending on the order in which they were first received).

However, the class does not include more adaptive strategies, such as sending the token received least often in past rounds.

Fix parameters k and T. For randomized and deterministic limited-history algorithms, we show the following lower bounds against an adaptive adversary:

[5]The mapping is a measure over the σ-algebra of cylinder sets, $\{\mathrm{Cyl}(x) \mid x \in (X_\perp)^*\}$, where $\mathrm{Cyl}(x) = \{y \in (X_\perp)^\omega \mid x \text{ is a prefix of } y\}$. It satisfies the usual constraints: e.g., the measures of all sequences $x_1, \ldots, x_m$ of length m add up to 1 (for any m), and the measure of a sequence $x_1, \ldots, x_m$ is at least as large as that of any sequence $x_1, \ldots, x_{m+1}$ that extends it by one round. However, for our purposes here we do not require the technical details.

[6]Technically, node u_0 sends a sequence of messages $(x_1, y_1), \ldots, (x_m, y_m)$, but here we care only about the tokens $x_1, \ldots, x_m$.

Theorem 3.40. *Any randomized limited-history k-token dissemination algorithm requires $\Omega(n + nk/T)$ rounds in the worst case to succeed with probability $1/2$ in T-interval connected networks.*

Theorem 3.40 asserts the existence of a "hard input assignment", but this assignment is somewhat "centralized": one node initially has all the tokens. For deterministic algorithms that use no control information, the lower bound also applies to input assignments where each node starts with exactly one token:

Theorem 3.41. *If the UID space is sufficiently large compared to the size of the network ($\Omega(n^3/T)$ suffices), then any deterministic algorithm that uses no control information requires $\Omega(n + n^2/T)$ rounds to solve all-to-all token dissemination in T-interval connected networks.*

We begin with a high-level overview, and then describe the lower bound in detail.

High-level overview. Fix a limited-history algorithm $\mathcal{A}$, a set of nodes $u_0, \ldots, u_{n-1}$, and an input assignment I where node u_0 receives all tokens and all other nodes receive at most one token each. Let $t_1 := (n-2)(k-1)/(4T)$. By Definition 3.39 there is a fixed distribution $\mathrm{D} : (X_\perp)^* \to [0, 1]$ governing u_0's output in all rounds, regardless of the dynamic graph we choose. Since $t_1 < (n-2)(k-1)/(4T)$, we can show that there exists a token x that is initially not known to at least $n-2$ nodes, such that with high probability node u_0 sends x fewer than $n-2$ times by time t_1. The idea in the proof is to always use u_0 as a buffer between the nodes that have already learned x and those that have not; since u_0 broadcasts x infrequently with high probability, in this manner we can limit the number of nodes that learn x. To preserve T-interval connectivity, whenever u_0 does broadcast x we must wait T rounds before we can do "damage control" and remove the nodes that have learned x from the vicinity of the nodes that have not. However, we will design the graph in such a way that only $O(T)$ nodes learn x while we wait to change the graph, so that the cost incurred every time u_0 broadcasts x is $O(T)$ nodes. To do this we keep the nodes that do not know x in a ring, while the nodes that do know x (or may know x) are connected in a clique (see Fig. 3-1). The only connection between the ring and the clique is through node u_0. Thus, token x can be learned by ring nodes only when u_0 sends it. To maintain this property we must occasionally "clean up" the graph and remove nodes that may have learned x from the ring. We show that we can do this in a manner that respects T-interval connectivity.

The technical details. Fix a set of n nodes $u_0, \ldots, u_{n-1}$. For a finite sequence $x_1, \ldots, x_m \in (X_\perp)^m$ and a token $x \in X$, let $\#x(x_1, \ldots, x_m)$ be the number of times x appears in the sequence. We treat $\#x$ as a random variable (over sequences of some fixed length m) and omit the sequence $(x_1, \ldots, x_m)$ where it is clear from the context.

We first show that we can (adaptively) construct a dynamic graph such that if node u_0 sends some token $x \in X$ infrequently, then this token does not propagate everywhere and the algorithm cannot terminate. Recall from Chapter 2 that an *adaptive adversary* constructs the dynamic graph on the fly, basing its choices in round r on the entire history up to to time $r-1$, including all random choices made in rounds $1, \ldots, r-1$. Technically, an adaptive adversary is a function that maps the history up to time $r-1$ (inclusive) to a set of edges for round r. In our case, the adversary's behavior depends only on the tokens sent by node u_0 in prior rounds.

Lemma 3.42. *Fix a token $x \in X$ that is initially known to node u_0 and at most one other node. There is an adaptive adversary $\mathcal{G}$ that constructs a T-interval connected dynamic graph, depending only on the tokens sent by node u_0, such that if u_0 sends tokens $x_1, \ldots, x_t$ in rounds $1, \ldots, t$, then at most $2T \cdot \#x(x_1, \ldots, x_t) + 2$ nodes know token x at time t.*

Proof. Let us describe the graph constructed by the adaptive adversary $\mathcal{G}$.

Execution segments. We divide the rounds $1, \ldots, t$ into segments $\alpha_1, \ldots, \alpha_m$; the graph remains static during each segment, but changes between segments. There are two types of segments in our construction.

- *Quiet* segments, each of which ends immediately after the first time that u_0 sends x in the segment (or at time t, whichever occurs sooner).
- *Active* segments, which last exactly T rounds each (or until time t if it occurs less than T rounds after the beginning of the segment).

The type (and hence also the duration) of each segment $\alpha_1, \ldots, \alpha_m$ is defined recursively. The first segment, α_1, is a quiet segment (i.e., it ends immediately after u_0 sends x for the first time). For each $i \geq 1$, if node u_0 sends token x at any point during segment α_i, then segment α_{i+1} is active; otherwise segment α_{i+1} is quiet. Note that the type of each segment depends only on the behavior of u_0 in the rounds that strictly precede the segment, so an adaptive adversary can already determine the segment type in the first round of each segment.

We say that a segment α_i is *even* (resp. *odd*) if the number of active segments that precede α_i is odd (resp. even). In particular, the first segment α_1 is even.

Construction of the dynamic graph. In each segment α_i we partition the nodes $\{u_0, u_1, \ldots, u_{n-1}\}$ into two sets, C_i and D_i. The nodes in C_i are referred to as *clean nodes*: they do not know token x. The nodes in D_i are *dirty nodes*: they may or may not know token x, but in the construction we pessimistically assume that they do. If at some point in the construction we have $D_i = V$, that is, all nodes are dirty, then the construction terminates and the graph remains fixed from that point on.

By our assumption on x, initially token x is known to node u_0 and possibly to one other node u_ℓ. Accordingly we define $D_1 = \{u_0, u_\ell\}$ and $C_1 = V \setminus \{u_0, u_\ell\}$. Let $v_1, \ldots, v_{n-2}$ be some arbitrary ordering of the nodes in C_1. In each segment i, the nodes of C_i are a contiguous subset $v_{L_i}, \ldots, v_{R_i} \subseteq C_1$, where $1 \leq L_i \leq L_{i+1} \leq R_{i+1} \leq R_i \leq n-2$ for all i. In the first segment α_i we have $L_i = 1$ and $R_i = n-1$. In subsequent segments α_{i+1} for $i \geq 1$, we define L_{i+1} and R_{i+1} recursively as follows:

- If α_i is a quiet segment, then $L_{i+1} := L_i$ and $R_{i+1} = R_i$.

- If α_i is active and α_{i+1} is an even segment, then $L_{i+1} := L_i + 2T$ and $R_{i+1} := R_i$.
- If α_i is active and α_{i+1} is an odd segment, then $L_{i+1} := L_i$ and $R_{i+1} := R_i - 2T$.

The edges of the graph are defined as follows (see Fig. 3-1): if r is a round during segment α_i, then

- If α_i is a quiet segment, then C_i is a line, with u_0 connected to either v_{L_i} or v_{R_i}, depending on whether the segment is even or odd. Formally,

  $$E(r) := (D_i \times D_i) \cup \{\{v_j, v_{j+1}\} \mid L_i \leq j < R_i\} \cup \{\{u_0, w_i\}\},$$

  where

  $$w_i := \begin{cases} v_{L_i} & \text{if the current segment is even, and} \\ v_{R_i} & \text{otherwise.} \end{cases} \tag{3.7.1}$$

- If α_i is an active segment, then C_i is a ring, with u_0 between nodes v_{L_i} and v_{R_i}. Formally,

  $$E(r) := (D_i \times D_i) \cup \{\{v_j, v_{j+1}\} \mid L_i \leq j < R_i\} \cup \{\{u_0, v_{L_i}\}, \{u_0, v_{R_i}\}\}.$$

Analysis of the execution. To unify the discussion of odd and even segments, let us define the following sets of nodes:

- The *red nodes*, red_i, are nodes $v_{L_i}, \ldots, v_{L_i+2T}$ if α_i is an even active segment, or nodes $v_{R_i-2T}, \ldots, v_{R_i}$ if α_i is an odd active segment.
- The *yellow nodes*, yellow_i, are nodes $v_{R_i-T}, \ldots, v_{R_i}$ if α_i is an even active segment, or nodes $v_{L_i}, \ldots, v_{L_i+T}$ if α_i is an odd active segment.
- The *green nodes*, green_i, during an active phase α_i are all the nodes in C_i that are neither red nor yellow. During a quiet phase, the green nodes are all nodes of C_i except node w_i.

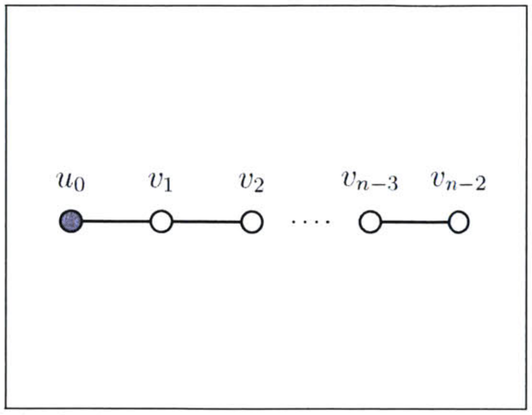
Informally, in terms of red, yellow and green nodes, the construction of the graph can be re-phrased as follows: after each active segment we remove the red nodes from the ring and move them into the clique; then, if the new segment is also active, the nodes that were previously yellow become red, while the T nodes adjacent to u_0 that were previously green now become yellow.

Our construction maintains the following invariant:

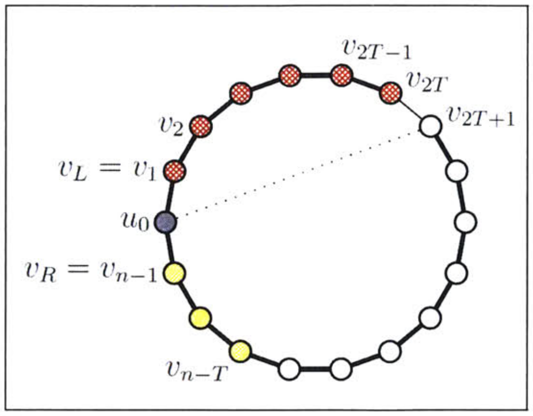
Invariant 3.43. Throughout the execution,

(a) After each round of a quiet segment α_i except the last round, all nodes of C_i do not know token x. In the last round only node $w_i \in C_i$ learns token x.

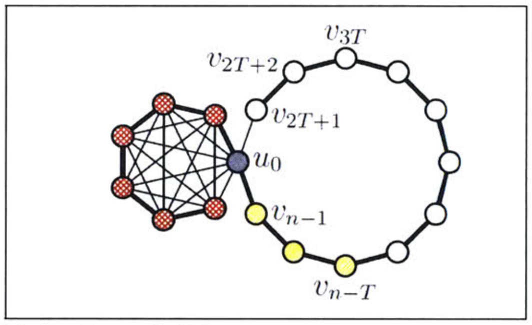
(b) At the beginning of each active segment α_i, the only nodes in C_i that may know token x are the T red nodes in red_i closest to u_0.



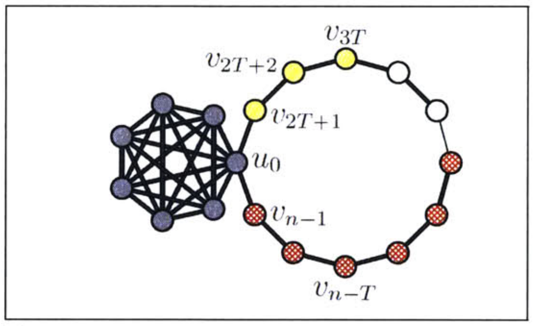
(a) The beginning of the execution, showing only nodes of C_1 (all but at most one node of V that receives token x in its input).



(b) The beginning of the first active segment. The dotted line indicates the edge that will be added at the end of the segment if the next segment is also active.



(c) At the end of the active segment, we remove the red nodes from the ring and place them in the clique. If the next segment is active, the ring is repaired by connecting u_0 to v_{2T+1}.



(d) In the second active segment, the nodes that were previously yellow become red, and the green nodes on u_0's other side become yellow.

Figure 3-1: Illustration for Lemma 3.45, with $T = 3$. Bold lines indicate edges that remain stable throughout the current segment and the next one. Green nodes are shown in white, yellow nodes in diagonal yellow lines, red nodes in red crosshatch, and nodes in D_i in solid grey.

(c) At the end of each active segment α_i, all green nodes (green_i) do not know token x. Moreover, if u_0 did not send token x during segment α_i, then the yellow nodes yellow_i also do not know token x.

In particular, the invariant implies that if a node u that did not initially know x learns x in some segment α_i, then either u is a red node in α_i, or u is a yellow node in α_i and will be a red node in α_{i+1}. Thus the number of nodes that know x at time t is bounded by the number of nodes that were red at any point before time t.

We prove the invariant by induction on the segment number. The first segment, α_1, is quiet. By choice of the token x and the set C_1, initially all nodes in C_1 do not know token x, and since u_0 does not send token x until the last round of α_1, the nodes of C_1 do not learn x until the last round of α_1. Finally, in the last round of α_i node u_0 sends x, but the only node that receives x is node w_1, which is u_0's only neighbor in C_i.

For the induction step, suppose that the invariant holds until the end of segment α_i, and consider segment α_{i+1}. Let us show parts (a)–(c) of the invariant:

(a) If α_{i+1} is a quiet segment, then by definition, node u_0 did not send token x at any point during segment α_i. By the induction hypothesis, the only nodes in C_i that know token x at the end of α_i are the red nodes red_i. However, these are removed from the ring: by definition of C_{i+1}, it contains only nodes that were green or yellow in segment α_i. Therefore the invariant holds at the beginning of α_i. Since u_0 does not send token x until the last round of α_i, the invariant continues to hold until the end (as we showed for α_1).

(b) If α_{i+1} is an active segment and α_i is quiet, the claim follows immediately from part (a) of the invariant (together with the fact that node w_i is red in α_{i+1}). If α_{i+1} is an active segment and α_i is also active, then the red nodes in α_{i+1} are the yellow nodes of α_i, plus their T neighbors in the direction away from u_0 on the ring. By the induction hypothesis, only the T nodes nearest to u_0 may know token x, so the invariant holds.

(c) By part (b), at the beginning of the current active segment only the T red nodes nearest to u_0 can know token x. If u_0 does not send x during the segment, then since the segment lasts T rounds, by the end of the segment token x can spread only to the next T nodes of the ring, i.e., only to the remaining red nodes. In this case the green and the yellow nodes do not know x at the end of the segment. On the other hand, if u_0 sends x, then the yellow nodes may also learn x, since they are the T nodes adjacent to node u_0 on the other side from the red nodes. But in this case the green nodes still do not know token x, so the invariant holds.

This concludes the proof of the invariant.

We have given an adaptive adversary $\mathcal{G}$ that maps a sequence $x_1, \ldots, x_t$ of tokens sent by node u_0 to a graph $\mathcal{G}(x_1, \ldots, x_t)$. From the invariant we showed above, it follows that in each such graph $\mathcal{G}(x_1, \ldots, x_t)$, a node learns token x in segment α_i only if it is a red node in segment α_i or α_{i+1}. (In particular, yellow nodes either do not learn x and go back to being green in the next segment, or they become red in the next active segment.) Since there are only $2T$ red nodes in every active segment, the number of nodes that learn token x is bounded by $2T$ times the number of active segments, which is itself bounded by $2T$ times the number of times u_0 sends token x, that is, by $2T \cdot \#x(x_1, \ldots, x_t)$. We assumed that initially at most two nodes know token x; therefore, by time t, at most $2T \cdot \#x(x_1, \ldots, x_t) + 2$ nodes know x, as desired. (If $2T \cdot \#x(x_1, \ldots, x_t) + 2 \geq n$, then the construction may terminate before time t, because we may run out of clean nodes and have $D_i = V$. However, in this case the lemma holds trivially.)

Before concluding the proof of the lemma, we must also show that the dynamic graph we constructed is T-interval connected. Observe that for even active segments, the line $\ell_{\text{even}} = v_{L_i+2T}, v_{L_i+2T-1}, \ldots, u_0, v_{R_i}, v_{R_i-1}, \ldots, v_{L_i+2T+1}$ over the nodes of C_i persists throughout the entire even active segment and the segment that follows it; similarly, for odd active segments, the line $\ell_{\text{odd}} = v_{R_i-2T}, v_{R_i-2T+1}, \ldots,$

$u_0, v_{L_i}, v_{L_i+1}, \ldots, v_{R_i-2T-1}$ persists throughout the segment and the segment that follows it. Also, the edges of D_i are stable once they appear. So we have the following types of intervals of T rounds:

- Intervals entirely contained in some segment: the graph is static and connected throughout the segment.

- Intervals beginning in some quiet segment and ending in the following active segment: no edges are removed at the end of the quiet segment, so the graph during the quiet segment persists into the next segment.

- Intervals beginning in some active segment α_i, and ending in the following segment (active or quiet): connectivity for the nodes of C_i is preserved along either ℓ_{even} or ℓ_{odd}; for the nodes of D_i connectivity is preserved along the clique edges; and node u_0 is stably connected through both $\ell_{\text{even}}/\ell_{\text{odd}}$ and the clique over D_i.

- Intervals beginning in some active segment, containing the next quiet segment, and ending in the next active segment: since we do not remove edges in the transition from a quiet segment to an active one, the previous case also applies here.

- Intervals containing an entire active segment but not contained in that segment: there are no such T-round intervals, because an active segment lasts T rounds.

$\square$

Lemma 3.42 does not involve any probabilistic reasoning — it shows that we can (adaptively and deterministically) construct a dynamic graph where if u_0 sends token x infrequently, then the algorithm cannot terminate. We now show that for limited-history algorithms that do not run for enough rounds, there is a token that u_0 sends infrequently with high probability: the following easy lemma shows that for any distribution over token sequences, roughly nk/T rounds are required to guarantee that each token is sent roughly n/T times with high probability.

Lemma 3.44. *Let I be an input assignment containing a total of k tokens $X = \{x_0, \ldots, x_{k-1}\}$, where node u_0 receives all k tokens, and for all $i \geq 1$ we have $\{x_0\} \subseteq I(u_i) \subseteq \{x_0, x_i\}$. Let*

$$t_1 := \frac{(n-2)(k-1)}{4T} - 1,$$

and let $\mathcal{D} : (X_\perp)^{t_1} \to [0,1]$ be a distribution over token sequences of length t_1. Then there is a token $x \in X$ such that

$$\Pr_{(x_1,\ldots,x_{t_1})\sim\mathcal{D}}\left[\#x(x_1, \ldots, x_{t_1}) < \frac{n-2}{2T}\right] \geq 1/2.$$

Proof. We have

$$\begin{aligned}\sum_{x\in X} \mathbb{E}\left[\#x\right] &= \sum_{x\in X}\sum_{r=1}^{t_1} \Pr_{\mathcal{D}}\left[x_r = x\right] \\ &= \sum_{r=1}^{t_1}\sum_{x\in X} \Pr_{\mathcal{D}}\left[x_r = x\right] = \sum_{r=1}^{t_1} 1 = t_1 < (n-2)(k-1)/(4T).\end{aligned}$$

It follows that there exist two tokens $x \neq x'$ such that $\mathbb{E}\left[\#x\right], \mathbb{E}\left[\#x'\right] < (n-2)/(4T)$. At least one of these tokens is not x_0; assume without loss of generality that $x \neq x_0$. Since $\mathbb{E}\left[\#x\right] < (n-2)/(4T)$, Markov's inequalty shows that

$$\Pr\left[\#x \geq \frac{n-2}{2T}\right] \leq \frac{\mathbb{E}\left[\#x\right]}{\frac{n-2}{2T}} < \frac{1}{2},$$

and the claim follows. □

Now we can combine Lemmas 3.44 and 3.42 to obtain the following:

Lemma 3.45. *Let I be an input assignment as in Lemma 3.44. For any limited-history algorithm, there is an adaptive adversary such that under input assignment I, the algorithm fails to solve k-token dissemination with probability $1/2$ by time*

$$t_1 := \frac{(n-2)(k-1)}{4T} - 1.$$

.

Proof. Since node u_0 initially receives all tokens, by Definition 3.39 there is a fixed mapping $\mathcal{D} : (X_\perp)^* \to [0,1]$ governing u_0's output in all future rounds, regardless of the dynamic graph. In the sequel, probabilities and expectations (over node u_0's output) are taken with respect to $\mathcal{D}$.

By Lemma 3.44, there is a token $x \in X$ such that

$$\Pr\left[\#x < \frac{n-1}{2T}\right] \geq \frac{1}{2}.$$

By Lemma 3.42, there is an adaptive adversary $\mathcal{G}$ that maps the sequence $y_1, y_2, \ldots, y_{t_1}$ of tokens sent by node u_0 in rounds $1, \ldots, t_1$ to a dynamic graph $\mathcal{G}(y_1, \ldots, y_{t_1})$, such that at time t_1 at most $2T \cdot \#x(y_1, \ldots, y_{t_1}) + 2$ nodes know token x. Since we started with at least $n-2$ nodes that do not know token x, whenever $2T \cdot \#x(y_1, \ldots, y_{t_1}) < n-2$, there is some node that does not know token x at time t_1 in $\mathcal{G}(y_1, \ldots, y_{t_1})$ and $\mathcal{A}$ cannot terminate.

By choice of x we have $\Pr\left[\#x < (n-2)/(2T)\right] \geq 1/2$, so $\Pr\left[2T \cdot \#x < n-2\right] \geq$

1/2. Thus,

$$\begin{aligned}
&\Pr[\mathcal{A} \text{ does not terminate by time } t_1] \\
&= \sum_{y_1,\ldots,y_{t_1} \in (X_\perp)^{t_1}} \Pr[y_1,\ldots,y_{t_1}] \cdot [\![\, \mathcal{A} \text{ does not terminate by } t_1 \text{ in } \mathcal{G}(y_1,\ldots,y_{t_1})]\!] \\
&\geq \Pr[2T \cdot \#x(y_1,\ldots,y_{t_1}) < n-2] \geq 1/2,
\end{aligned}$$

where here the notation $[\![P]\!]$ denotes the value of Boolean predicate P and the sequence $y_1,\ldots,y_{t_1}$ is drawn from the distribution $\mathcal{D}$ governing node u_0's output. □

From Lemma 3.45 we obtain our two lower bounds:

Proof of Theorem 3.40. A lower bound of $\Omega(n)$ is demonstrated trivially in a static line network where at least one token starts at one end of the line: $n-1$ rounds are required for the node at the other end of the line to receive the token. Therefore we focus on showing the bound of $\Omega(nk/T)$. By Lemma 3.45, there is no limited-history algorithm that succeeds with probability greater than 1/2 by time

$$t_1 = \frac{(n-2)(k-1)}{4T} = \Omega\left(\frac{nk}{T}\right).$$

The claim follows. □

Proof of Theorem 3.41. As in Theorem 3.40, we focus on the $\Omega(n^2/T)$ lower bound. Let the UID space also be the token domain. We rely on Lemma 3.45 again. We now start from an input assignment where each node u_i receives one unique token u_i, and we show that either

1. The algorithm runs for $\Omega(n^2/T)$ rounds, or .

2. We reach time t_0 in which some node, without loss of generality we assume that it is node u_0, knows all the tokens, and for all $i \neq 1$ we have $A_{u_i}(t_0) \subseteq \{u_0, u_i\}$. Here, as usual, $A_u(t)$ stands for the set of tokens received by node u no later than time t.

In case (2), we then apply Lemma 3.45, starting from the configuration at time t_0. Note that this configuration satisfies the requirements on the input assignment in Lemma 3.45 (it is easy to verify that the lemma applies starting from any time, not just time 0).

Let us now show that one of the two cases above must hold. Assume that $|U| \geq cn^3/T + n - 1$ for some constant $c \geq 0$. Say that a node $u \in \mathcal{U}$ *fires in round* r if when node u receives as its input the set $\{u\}$, and hears nothing in the first $r-1$ rounds of the execution, node u will stay silent in rounds $1,\ldots,r-1$ and then spontaneously send its token in round r. If u does not fire in any round $r' \leq r$ we say that u is *passive until round* r.

We divide into two cases.

Case I. There exist $u_0, \dots, u_{n-1} \in \mathcal{U}$ that are all passive until round cn^2/T. In this case we construct the static clique over $u_0, \dots, u_{n-1}$, give each node its own UID as its input, and let the algorithm run. During the first cn^2/T rounds, all nodes send nothing, and no node learns new tokens. Consequently the algorithm cannot terminate by round $cn^2/T = \Omega(n^2/T)$.

Case II. All but at most $n - 1$ nodes in $\mathcal{U}$ fire no later than round cn^2/T. Since $|\mathcal{U}| \geq cn^3/T + n - 1$, by the pigeonhole principle there must exist a round $r_0 \leq cn^2/T$ such that at least n nodes fire in round r_0. Let $u_0, \dots, u_{n-1}$ be n such nodes, and let S be the static star with u_0 at the center: $S = (V, E_S)$, where $E_S(r) = \{\{u_0, u_i\} \mid i > 1\}$ for all $r \leq r_0$.

Because all nodes fire in round r_0, when the algorithm is executed in S, the network is silent until round r_0, and in round r_0 all nodes send their tokens. At time r_0 (i.e., after round r_0), node u_0 knows all tokens, and the remaining nodes know token u_0 and the token they started with, that is, their own UID. At this point we apply Lemma 3.45 (with $k = n$), which shows that $\Omega(n^2/T)$ more rounds are necessary after time r_0 to solve all-to-all token dissemination. (The statement of Lemma 3.45 concerns randomized algorithms, but of course, it applies to deterministic algorithms as well.) □

3.8 Summary and Open Problems

In this chapter we focused on two problems: token dissemination and counting. We now summarize our results, and point out several open problems.

Token dissemination. We showed that it is possible to deterministically solve token dissemination (and therefore also counting) in $O(n^2/T + n)$ rounds in T-interval connected graphs. We gave a lower bound of $\Omega(n^2/T + n)$ rounds for a restricted class of algorithms; however, the deterministic (and randomized) round complexity of token dissemination in T-interval connected graphs for general algorithms remains an open problem. A nearly-tight lower bound of $\Omega(n^2/\log n)$ is given in [43] for 1-interval connected graphs, and this was later strengthened in [68] to a bound of $\tilde{\Omega}(n + n^2/T^2)$ for T-interval connected graphs. Both bounds hold for *centralized token-forwarding algorithms*, but there remains a factor of T between the best known upper bound and lower bound.

Oblivious vs. adaptive adversaries. Our $O(n^2/T + n)$-round token dissemination algorithm from Section 3.4.3 is deterministic; for deterministic algorithms there is no distinction between an oblivious adversary and an adaptive one. However, the randomized $\Omega(n^2/T + n)$ lower bound we gave in Section 3.7 requires the adversary to be adaptive. The question of whether randomization allows us to improve on the running time of $O(n^2/T + n)$ when the adversary is oblivious remains an interesting open problem.

Algorithms that combine tokens. Our results focused on *token-forwarding algorithms*, which treat the tokens as black boxes. We assumed throughout that each message can contain a constant number of tokens, or polylog(n) bits. The results extend to the case where $d > 1$ tokens can be packed into each message, and we obtain a speedup linear in d.

Following the publication of the results in this chapter, it was shown in [67] that using *random linear network coding*, one can improve the running time from $O(n^2/d)$ rounds to $O(n^2/d^2)$ in 1-interval connected graphs. For the case where only $O(\log n)$ bits can be sent in each message, [67] gives an $O(n^2/\log n)$-round algorithm, a logarithmic improvement over our results.

It is also shown in [67] that when the dynamic graph satisfies a constraint called *T-stability*, which asserts that the graph changes only once every T rounds, token dissemination can be solved in $O(n^2/(d^2T^2))$ rounds, again improving upon the linear $O(n^2/(dT))$ improvement our algorithms obtain in this case. Strictly speaking, the results from [67] are incomparable with ours, because T-stability and T-interval connectivity are incomparable: T-stability does not allow any changes to the graph, except once every T rounds; when the graph changes, it may change completely, and bear no relationship to the graph in the previous T rounds (in particular the intersection of the new graph and the old one does not have to be connected). In contrast, T-interval connectivity allows the graph to change in every round, but requires a connected subgraph to persist over a "sliding window" of T rounds, so there can be no sharp transitions. We can define a weaker constraint, let us call it *weak T-interval connectivity*, that divides the execution into consecutive non-overlapping blocks of T rounds each, and requires the existence of a stable connected subgraph in each block of T rounds. Weak T-interval connectivity allows sharp transitions between blocks; it is strictly weaker than both T-stability and T-interval connectivity. All the algorithms we gave in this chapter, with the exception of Section 3.5.2, work under weak T-interval connectivity.

Many problems remain open. The first and most immediate is to extend the results of [67] to T-interval connected graphs (or show that the results do not extend). A larger problem is to find non-trivial lower bounds on the round complexity of algorithms that may combine tokens. This is likely to be a challenge, as this problem is currently not well understood even for static networks.

Chapter 4

Consensus and Coordination in Dynamic Networks

In this chapter we study *consensus*, a fundamental distributed problem, in the context of dynamic networks. In the classical formulation of consensus, each node receives a binary input, and the goal is for each node to produce a binary output satisfying the following conditions:

- Agreement: all nodes produce the same output.
- Validity: the output produced by the nodes is the input to some node.

One way to solve consensus is to have all nodes agree on the minimum input (see, e.g., the FLOODMIN algorithm in [110]). In Section 4.2 we show that this strategy is optimal: although consensus appears on first sight to be "easier" than minimum (in the sense that it is less constrained), in the absence of an *a priori* bound on the size of the network consensus is as hard as computing the minimum.

In addition to "standard" consensus, which requires nodes to agree on an *output value*, we also study variants of the problem that require nodes to agree on a *time*. In Section 4.3 we study *simultaneous consensus*, a variant of consensus where all nodes must produce their outputs at the same time. We give a lower bound on simultaneous consensus and any other "non-trivial" simultaneous task, and show that any such task is at least as hard as computing an upper bound on the size of the network. Next, in Sections 4.4 and 4.5 we turn our attention to Δ-*coordinated consensus*, a weaker version of coordinated consensus which requires nodes to output their values within Δ rounds of each other, and give upper and lower bounds on this problem.

One of the main themes of this chapter is the connection between *coordination*, *causality* and *knowledge*. This connection, formalized through *epistemic logic*, was already known for traditional network models in which consensus has been studied in the past — namely, single-hop networks with various types of faults (we refer to [48] for a comprehensive account of these results). We extend the connection to multi-hop networks, networks that are not known in advance to the participants, and dynamic networks, and we show that it yields strong lower bounds and reductions from problems that on first sight appear harder than consensus. Some of our results (in Sections 4.2 and 4.4) apply to static networks as well as dynamic ones. To our knowledge, this is the first study of consensus, knowledge and common knowledge in general multi-hop networks.

In the previous chapter we assumed that nodes can send only a limited amount of information in each message. In contrast, in this chapter we study *full-information protocols*, where each node sends its entire history in each message (see Section 4.1 for a more precise definition). Full-information protocols are interesting because they extract the maximum amount of information out of the system: if it is at all possible for a node to know some fact at some point in the execution, then it will know it in a full-information protocol. A lower bound against full-information protocols must rely only on the inherent uncertainty of the distributed system; it cannot appeal to "information bottleneck" arguments such as the one in Section 3.7 and the ones we will see in Chapter 6. We have already seen simple full-information lower bounds in Section 3.1 (Lemma 3.6 and Proposition 3.7). In this chapter we will give more involved lower bounds and explore their underpinnings in epistemic logic.

4.1 Epistemic Logic in Distributed Computing

In this section we review several basic concepts in epistemic logic. The concept of *knowledge* was already introduced in Section 2.4, where we gave an operational description of what it means for a node to "know" a fact (Definition 2.25). We now formalize knowledge as a modal operator of epistemic logic and relate it to several other operators. The definitions in this section follow the conventions of [48], with several minor changes necessary to to deal with the fact that in our model the set of participants in an execution is not fixed in advance.

Executions and timepoints. Recall from Section 2.4 that knowledge is studied with respect to a set $\mathcal{C}$ of possible executions induced by a given algorithm. We represent each execution in $\mathcal{C}$ as a pair (G, I), where G is a dynamic graph and I is an input assignment that maps nodes to inputs. A *timepoint* (also called a *point* or *configuration*) of the system is a pair $(\alpha, t) \in \mathcal{C} \times \mathbb{N}$, represented by a vector of the states of the nodes in the system at time t in α. We let $V(\alpha)$ denote the set of nodes participating in α.

Full-information protocols. A *full-information protocol* is a protocol in which nodes store their entire history in their local state, and send their entire state in each message. We refer to [48] for the details of the construction. For our purposes here, a full-information protocol is a protocol that achieves the following: for any two executions α, β, node $u \in V(\alpha) \cap V(\beta)$, and time t,

$$(\alpha, t) \sim_u (\beta, t) \qquad \Leftrightarrow \qquad \alpha \text{ and } \beta \text{ are } (v, t')\text{-identical for all } (v, t') \in \mathsf{past}(u, t)_\alpha.$$

Recall from Definition 2.27 that two executions α, β are said to be *(u, t)-identical* if node u participates in both executions, and has the same input and edges in every round up to time t in both executions. A full-information protocol is therefore a protocol that allows nodes to distinguish between two executions whenever they differ in any way that causally influences the node. We saw in Proposition 2.28 that the other direction, which asserts that if α, β are identical on $\mathsf{past}(u,t)$ then u cannot distinguish them at time t, holds for *any* protocol. Thus, a full-information protocol extracts as much information as can possibly be extracted about an execution.

In the rest of the chapter, unless stated otherwise, we assume a full-information protocol. Our lower bounds will rely heavily on Proposition 2.28 to bound the power of such protocols.

The similarity graph. Fix a set $\mathcal{C}$ of executions. We let $(\alpha,t) \sim_u (\beta,t)$ denote the fact that node u participates in executions $\alpha, \beta \in \mathcal{C}$ and has the same state at time t in α and β, that is, node u *cannot distinguish* between (α,t) and (β,t).[1] The relations $\{\sim_u \mid u \in \mathcal{U}\}$ induce an undirected, edge-labeled graph on $\mathcal{C} \times \mathbb{N}$. This graph is called the *similarity graph.* We are interested in the connected components of the similarity graph: we let $(\alpha,t) \sim (\beta,t)$ denote the fact that (α,t) and (β,t) are in the same component of the similarity graph, that is, either $\alpha = \beta$ or there exist $u_1, \ldots, u_k \in \mathcal{U}$ and executions $\alpha_1 = \alpha, \alpha_2, \ldots, \alpha_{k+1} = \beta$ such that $(\alpha,t) = (\alpha_1,t) \sim_{u_1} (\alpha_2,t) \sim_{u_2} \ldots \sim_{u_k} (\alpha_{k+1},t) = (\beta,t)$. In this case we say that (α,t) and (β,t) are *similar.*

The following property of the similarity graph will be useful in the sequel:

Proposition 4.1. *If $(\alpha,t) \sim_u (\beta,t)$ and $t' < t$, then $(\alpha,t') \sim_u (\beta,t')$.*

Proof. If $(\alpha,t) \sim_u (\beta,t)$ then since we assume a full-information protocol, α and β are (v,s)-identical for all $(v,s) \in \mathsf{past}(u,t)_\alpha$. In particular, α and β are (v,s)-identical for all $(v,s) \in \mathsf{past}(u,t')_\alpha \subseteq \mathsf{past}(u,t)_\alpha$. It follows from Proposition 2.28 that $(\alpha,t') \sim_u (\beta,t')$. □

As an immediate consequence of Proposition 4.1, two executions α, β are similar at time t iff they are similar at all times $t' \leq t$.

Epistemic logic. Fix a set P of *atomic propositions*, that is, basic facts about which we wish to reason. The atomic propositions are interpreted by a *valuation* $v : \mathcal{C} \times \mathbb{N}^+ \times P \to \{0,1\}$, which maps each timepoint (α,t) and atomic proposition $p \in P$ to a truth value $v((\alpha,t),p) \in \{0,1\}$. If $v((\alpha,t),p) = 1$, then we denote $(\alpha,t) \models p$, and say that (α,t) *satisfies* p.

Each atomic proposition is a formula of epistemic logic. In addition, complex formulas can be built up from simpler formulas using the standard Boolean operators $(\wedge, \vee, \rightarrow, \neg)$, and the following *epistemic operators*:

[1]Note that the indistinguishability relation $\sim_u$ is defined only over points $(\alpha,t), (\beta,t)$ that share the same time t. In general (e.g., in [48]) this relation is usually defined over points $(\alpha,t), (\beta,s)$ at possibly differing times t, s. However, in this thesis we study only *synchronous systems,* where all nodes know the current time; in our model, if $t \neq s$ then any node u can distinguish point (α,t) from point (β,s). Thus our definition is equivalent to the more general one: in both definitions we can only have $(\alpha,t) \sim_u (\alpha,s)$ if $t = s$.

- *Knowledge*: if φ is a formula, then for each $u \in \mathcal{U}$, the formula $K_u\varphi$ is interpreted as follows:

$$(\alpha, t) \models K_u\varphi \quad \Leftrightarrow \quad [u \in V(\alpha) \wedge \forall(\beta, t) : (\alpha, t) \sim_u (\beta, t) \Rightarrow (\beta, t) \models \varphi] .$$

(This definition is a small generalization of Definition 2.25 from Section 2.4; Definition 2.25 addressed only the case where φ is an atomic proposition, but we now allow φ to be an arbitrary knowledge formula.)

- *Everyone knows*: for any formula φ, the formula $E\varphi$ is interpreted by

$$(\alpha, t) \models E\varphi \quad \Leftrightarrow \quad [\forall u \in V(\alpha) : (\alpha, t) \models K_u\varphi] .$$

- *Common knowledge*: if φ is a formula, then formula $C\varphi$ is interpreted as follows:

$$(\alpha, t) \models C\varphi \quad \Leftrightarrow \quad [\forall(\beta, t) \sim (\alpha, t) : (\beta, t) \models \varphi] .$$

Intuitively, a formula φ is common knowledge ($C\varphi$) if φ is true, everyone knows that φ is true, everyone knows that everyone knows that φ is true, and so on. This is because "everyone knows that... everyone knows φ" at nesting depth k corresponds to φ holding for all timepoints within distance k in the similarity graph:

Proposition 4.2 ([48]). *For all timepoints (α, t), formulas φ and $k \in \mathbb{N}$, $(\alpha, t) \models E^k\varphi$ iff $(\beta, t) \models \varphi$ for all (β, t) at distance at most k from (α, t) in the similarity graph.*

Proof. By induction on k. For $k = 0$ we have $E^0\varphi = \varphi$, so the claim holds trivially. For the inductive step, suppose that the claim holds for k, and consider $k + 1$. By definition, $(\alpha, t) \models E^{k+1}\varphi$ iff $(\alpha, t) \models K_u E^k\varphi$ for all $u \in V(\alpha)$. Unwinding the definitions further, this holds iff $(\beta, t) \models E^k\varphi$ for all (β, t) such that for some $u \in V(\alpha)$ we have $(\beta, t) \sim_u (\alpha, t)$. These are exactly the timepoints (β, t) at distance one from (α, t) in the similarity graph. (The edges of the similarity graph are defined with respect to all possible nodes in $\mathcal{U}$, and here we use only edges $\sim_u$ for $u \in V(\alpha)$; however, we can only have $(\alpha, t) \sim_v (\beta, t)$ if $v \in V(\alpha) \cap V(\beta)$, so the edges coincide.) By the induction hypothesis, for each such point we have $(\beta, t) \models E^k\varphi$ iff $(\gamma, t) \models \varphi$ for all (γ, t) at distance at most k from (β, t). Putting everything together we see that $(\alpha, t) \models E^{k+1}\varphi$ iff $(\gamma, t) \models \varphi$ for all (γ, t) at distance at most $k + 1$ from (α, t), as desired. $\square$

Corollary 4.3. *The formula $C\varphi$ is logically equivalent to $\bigwedge_{i=0}^{\infty} E^i\varphi$.*

Throughout Chapter 3 we used Lemma 2.21 as a "termination criterion", which allowed us to determine when nodes have heard from everyone by computing or approximating the size of their past sets. We now show that this termination criterion can be refined slightly to yield an optimal termination test. Let us first observe that Lemma 2.21 is easily generalized to yield:

Lemma 4.4. *For any times $t \leq t' \leq s$, if $\mathsf{past}(u,s)_t \neq V$ then $|\mathsf{past}(u,s)_t \setminus \mathsf{past}(u,s)_{t'}| \geq t' - t$.*

Proof. An easy induction on $t' - t$, as in Lemma 2.21. The base case ($t' = t$) is immediate. For the induction step, suppose that the claim holds for all nodes u and times $t \leq t' \leq s$ such that $t' - t = k$, and fix u and $t \leq t' \leq s$ such that $t' - t = k + 1$. Suppose further that $\mathsf{past}(u,s)_t \neq V$. By Proposition 2.12, $\mathsf{past}(u,s)_{t+1} \subseteq \mathsf{past}(u,s)_t$, so we also have $\mathsf{past}(u,s)_{t+1} \neq V$. From 1-interval connectivity, there is some edge (v,w) in the cut $(V \setminus \mathsf{past}(u,s)_{t+1}, \mathsf{past}(u,s)_{t+1})$, and the source v of this edge is in $\mathsf{past}(u,s)_t \setminus \mathsf{past}(u,s)_{t+1}$. Since $t' - (t+1) = k$, from the induction hypothesis we have $|\mathsf{past}(u,s)_{t+1} \setminus \mathsf{past}(u,s)_{t'}| \geq t' - (t+1)$, and therefore $|\mathsf{past}(u,s)_t \setminus \mathsf{past}(u,s)_{t'}| \geq t' - (t+1) + 1 = t' - t$. □

We can now refine our termination criterion and re-cast it in terms of epistemic logic. Let P_u be a propositional formula such that $(\alpha, t) \models P_u$ iff $\mathsf{past}(u,t)_{\alpha,0} = V(\alpha)$, that is, P_u holds iff u has heard from everyone.

Lemma 4.5. *The following conditions are equivalent:*

(a) $(\alpha, t) \models K_u P_u$,

(b) There exist times t', t'' such that $t'' \leq t' \leq t$ and $|\mathsf{past}(u,t)_{\alpha,t''} \setminus \mathsf{past}(u,t)_{\alpha,t'}| < t' - t''$,

(c) $|\mathsf{past}(u,t)_{\alpha,0}| = |\mathsf{past}(u,t)_{\alpha,1}|$.

Proof. We will show that (c) ⇒ (b) ⇒ (a) ⇒ (c).

Condition (c) immediately implies condition (b) (simply set $t' = 1, t'' = 0$). Condition (b) implies condition (a) because of Lemma 4.4: if there exist $t'' \leq t' \leq t$ such that $|\mathsf{past}(u,t)_{\alpha,t''} \setminus \mathsf{past}(u,t)_{\alpha,t'}| < t' - t''$, then by Proposition 2.12 and Lemma 4.4 we must have $\mathsf{past}(u,t)_{\alpha,0} \supseteq \mathsf{past}(u,t)_{t''} = V(\alpha)$. Since $|\mathsf{past}(u,t)_{\alpha,t''} \setminus \mathsf{past}(u,t)_{\alpha,t'}| < t' - t''$ always implies $\mathsf{past}(u,t)_{\alpha,0} = V(\alpha)$, whenever $|\mathsf{past}(u,t)_{\alpha,t''} \setminus \mathsf{past}(u,t)_{\alpha,t'}| < t' - t''$ node u *knows* that $\mathsf{past}(u,t)_{\alpha,0} = V(\alpha)$.[2]

It remains to show that condition (a) implies condition (c). We will show the contrapositive. Suppose that $|\mathsf{past}(u,t)_{\alpha,0}| \neq |\mathsf{past}(u,t)_{\alpha,1}|$. By Proposition 2.12 we have $\mathsf{past}(u,t)_{\alpha,1} \subseteq \mathsf{past}(u,t)_{\alpha,0}$, so there must exist some node $v \in \mathsf{past}(u,t)_{\alpha,0} \setminus \mathsf{past}(u,t)_{\alpha,1}$. We can construct another execution β just as we did in Lemma 3.6, where we add an arbitrary number of nodes to $V(\alpha)$ and connect them only to node v; the nodes of $V(\alpha) \setminus \{v\}$ have the same edges in β as they do in α, and all nodes of $V(\alpha)$ have the same input in α and β. By Proposition 2.28, $(\alpha, t) \sim_u (\beta, t)$, as α and β are identical on every time-node in $\mathsf{past}(u,t)_\alpha$ (recall that $(v,1) \notin \mathsf{past}(u,t)_\alpha$, and hence $(v,s) \notin \mathsf{past}(u,t)_\alpha$ for all $s \geq 1$ as well). Since $(\beta, t) \not\models P_u$, it follows that $(\alpha, t) \not\models K_u P_u$. □

[2]Epistemic logic is an instance of the modal logic S5 [48], and in particular it satisfies the following "distributivity" axiom: $K_u(\psi_1 \rightarrow \psi_2) \rightarrow (K_u\psi_1 \rightarrow K_u\psi_2)$. Here we are instantiating this axiom with ψ_1 = "$|\mathsf{past}(u,t)_{\alpha,t'} \setminus \mathsf{past}(u,t)_{\alpha,t''}| < t' - t''$" and $\psi_2 = P_u$. We also use the fact that since the protocol is full-information, whenever $|\mathsf{past}(u,t)_{\alpha,t'} \setminus \mathsf{past}(u,t)_{\alpha,t''}| < t' - t''$, node u *knows* that $|\mathsf{past}(u,t)_{\alpha,t'} \setminus \mathsf{past}(u,t)_{\alpha,t''}| < t' - t''$.

Lemma 4.5 gives us an HF_n algorithm that is optimal in every execution: the HF_n task is the problem of determining when $\mathsf{past}(u,t)_0 = V$, that is, the problem of determining when P_u holds. Lemma 4.5 shows that node u can know that P_u holds only when condition (c) holds. Thus an optimal way to solve HF_n is to run a full-information protocol, evaluate condition (c) at every timestep, and halt as soon as the condition is satisfied. In the next section we will show that consensus is as hard as HF_n in the worst case, so this also yields a protocol for consensus that is worst-case optimal. However, all-case optimality is lost for consensus. (This is unavoidable, as no all-case optimal algorithm for consensus exists [48].)

4.2 Consensus and Causality

In this section we study the consensus task with no timing constraints. Recall that in consensus, the only constraint on the global decision value is validity: the decision value must be the input to some node. This appears to be a fairly weak constraint. However, we now show that even though consensus is not a deterministic function of the nodes' inputs, in the absence of an *a priori* count it is still as hard as HF_n, which means that it is no easier than computing a deterministic function such as the minimum input.

Lower bounds on consensus are often proven by a *partitioning argument*, where it is shown that due to a lack of communication, two parts of the network fail to agree on their outputs. Our analysis of the hardness of consensus relies on the following partitioning result:

Lemma 4.6. *Let $U, V \subseteq \mathcal{U}$ be disjoint sets of UIDs, and fix a class $\mathcal{C}$ of executions of a consensus protocol. Let $\alpha \in \mathcal{C}$ be an execution over the nodes of U where the input to all nodes is* 0, *and let $\beta \in \mathcal{C}$ be an execution over the nodes of V where the input to all nodes in* 1. *If $\mathcal{C}$ admits all executions over the nodes $U \cup V$, then in either α or β, no node halts before it has heard from everyone.*

Proof. Suppose for the sake of contradiction that there exist nodes $u \in U, v \in V$ such that node u halts at time t in α and node v halts at time t' in β, but $\mathsf{past}(u,t)_{\alpha,0} \neq U$ and $\mathsf{past}(v,t')_{\beta,0} \neq V$. Let $u' \in U \setminus \mathsf{past}(u,t)_{\alpha,0}$ and let $v' \in \mathsf{past}(v,t')_{\beta,0}$. We construct a third execution γ over the nodes $U \cup V$ as follows:

- In each round, the dynamic graph of γ contains the edges present in the corresponding round of α and β, and also the edge $\{u', v'\}$ (that is, we include both directed edges (u', v') and (v', u')).
- The input to the nodes of U is 0 and the input to the nodes of V is 1.

Executions α, γ are $(U \setminus \{u'\}, t)$-identical, and executions β, γ are $(V \setminus \{v'\}, t')$-identical. By Proposition 2.12, since $u' \notin \mathsf{past}(u,t)_{\alpha,0}$ we have $u' \notin \mathsf{past}(u,t)_{\alpha,s}$ for all $s \leq t$, so by Proposition 2.28, node u cannot distinguish α from γ and it halts at time t. In α node u must output 0, because the input to all nodes is 0; therefore in γ node u also halts and outputs 0 at time t. Similarly, node v cannot distinguish

β from γ at time t'; since in β node v must output 1, node v also outputs 1 in γ, violating agreement. $\square$

We can use Lemma 4.6 to show that consensus is as hard as HF_n. The reduction we use for this purpose does not quite fit Definition 2.5; it does not use the consensus algorithm as a black box. Nevertheless, we are able to convert any consensus algorithm into a HF_n algorithm with the same worst-case time complexity and the same asymptotic message complexity.

Corollary 4.7. *If there is no* a priori *upper bound on the size of the network, then any consensus algorithm $\mathcal{A}$ can be transformed into an HF_n algorithm $\mathcal{A}'$ that has the same running time and message size as $\mathcal{A}$ in any execution.*

Proof. The absence of an *a priori* bound on the size of the network corresponds to an infinite set of UIDs (see Example 2.26). Assume without loss of generality that the UID space is $\mathbb{N}$, and fix a consensus algorithm $\mathcal{A}$. One of the following cases holds:

I. For every set U of UIDs and every dynamic graph, with the all-zero input assignment, no node halts before hearing from everyone. Then we can solve HF_n by running $\mathcal{A}$ with input 0 at every node, and halting when $\mathcal{A}$ halts.

II. There exists a set U of UIDs and a dynamic graph G over U such that with the all-zero input assignment, some node halts before hearing from everyone. Let $f : \mathbb{N} \to \mathbb{N} \setminus U$ be some injective mapping (e.g., we can map every number m to $m + 1 + \max(U)$), and let $f(X)$ denote the image of f on a set $X \subseteq \mathbb{N}$. To solve HF_n, we execute $\mathcal{A}$ using input 1 at every node, but instead of using the original UIDs, every node u executes $\mathcal{A}$ with UID $f(u)$. By Lemma 4.6, for any dynamic graph $H = (V, E)$, when $\mathcal{A}$ is executed this way, no node halts before hearing from everyone (because $f(V)$ and U are disjoint by choice of f).

$\square$

Corollary 4.8. *If $\mathcal{A}$ is a consensus algorithm that uses no* a priori *bound on the size of the network, then there is an algorithm for computing the minimum input that has the same running time as $\mathcal{A}$ in every execution, and uses messages of size $M + d$, where M is the message size of $\mathcal{A}$ and d is the size of the input to each node.*

Proof. By Corollary 4.7, we can transform $\mathcal{A}$ into an algorithm $\mathcal{A}'$ for HF_n, which uses the same running time and message size as $\mathcal{A}$. To compute the minimum, we run $\mathcal{A}'$ and append the smallest input heard so far to every message. When $\mathcal{A}'$ halts we halt and output the minimum received so far. $\square$

When the UID space is finite, the size of this set is an upper bound on the size of the network, and we can no longer find an injective but non-surjective UID mapping f as we did in Corollary 4.7. (Indeed, the existence of an injective but non-surjective mapping from a set to itself is one way to define what an infinite set is.) To see the problem, consider the case where the "bad set" U is exactly $\mathcal{U}$, the entire UID space. Clearly we cannot "circumvent" U by mapping to a disjoint set of UIDs.

In fact, when the UID space is finite it is no longer true that nodes must implicitly solve HF_n if they are to solve consensus. For example, consider the following algorithm, which uses an upper bound of N on the size of the network: in each round, each node sends the UIDs of all nodes it has heard from so far, together with the input to each such node. (This is a simple variant of Algorithm 9, where we attach input values to UIDs.) If a node ever hears of a total of more than $N/2$ nodes that received the same input x, it immediately halts and decides x. Otherwise nodes use the usual termination test from Algorithm 9 to determine when they have heard from everyone, and at that point they halt and output the majority input. The algorithm is correct because all nodes always decide on the majority input value. This algorithm sometimes allows nodes to halt after hearing from only $N/2$ nodes, even when the network contains as many as N nodes. Thus, some nodes may halt before hearing from everyone.

Another "bad strategy" for our purposes is to always decide on the input of some special node, say node 1, if that node is present. In this case node 1 may decide without hearing from any other node, because it knows the decision value will be its own input. However, we can show that these two strategies — designating some "special nodes" or deciding on the majority value when the graph contains at least $N/2$ nodes — are essentially the only tricks that can be played: if the algorithm is comparison based (so no UIDs are "special"), all nodes must always hear from either all nodes in the graph or at least $N/2$ nodes, whichever is smaller.

Lemma 4.9. *If $\mathcal{A}$ is a comparison-based algorithm, where nodes nodes access UIDs in a black-box manner and can only compare them to each other, and N is the size of the UID space, then either for the all-zero input assignment or the all-one input assignment, $\mathcal{A}$ solves $HF_{\min\{n,N/2\}}$ in all executions.*

Proof. Suppose not. Then there is some dynamic graph $G_0 = (V_0, E_0)$ such that in execution $\alpha = (G_0, I_0)$ where I_0 is the all-zero input assignment, some node u_0 halts at time t_0 such that $|\mathsf{past}(u_0, t_0)_0| < \min\{|V_0|, N/2\}$, and similarly there is a graph $G_1 = (V_1, E_1)$ such that in execution β with the all-one input assignment some node u_1 halts at time t_1 with $|\mathsf{past}(u_1, t_1)_0| < \min\{|V_1|, N/2\}$.

Let U_0 be a set of $|\mathsf{past}(u_0, t_0)_0|$ UIDs, such that $U_0 \cap \mathsf{past}(u_1, t_1)_0 = \emptyset$. (We know that there exist sufficiently many UIDs for U_0, because by assumption $|\mathsf{past}(u_0, t_0)_0| < N/2$ and $|\mathsf{past}(u_1, t_1)_1| < N/2$, and we assumed that there are N UIDs. Therefore there exist $N - |\mathsf{past}(u_1, t_1)_0| > N/2 > |\mathsf{past}(u_0, t_0)_0|$ UIDs that are not in $\mathsf{past}(u_1, t_1)_0$.) Let f be an order-preserving mapping from V_0 to U_0 (i.e., f maps the i-th UID in V_0 to the i-th UID in U_0), and let H_0 be the dynamic graph obtained by applying the isomorphism f to G_0. Since the algorithm is comparison-based and f preserves the order of UIDs, the state of each node $f(v)$ at time s in H_0 is the same as the state of node v at time s in G_0. Therefore, if γ is the execution over H_0 with the all-zero assignment, $f(u_0)$ halts in γ at time t_0 with $|\mathsf{past}(f(u_0), t_0)_0| < \min\{|V_0|, N/2\}$. In particular, in β node u_1 halts before hearing from everyone, and in γ node $f(u_0)$ halts before hearing from everyone. This contradicts Lemma 4.6, which shows that since γ and β are over disjoint node sets and have the all-zero and all-one input assignment respectively, in at least one of the two executions no node can halt only after hearing from everyone.

□

4.3 Simultaneous Tasks and Common Knowledge

In static, fully-connected networks, it is well known that simultaneous actions are closely related to common knowledge [69, 48, 45, 113]. In this section we revisit this connection and explore its implications for coordination in dynamic networks.

4.3.1 Simultaneous Tasks

Recall that a task T is said to be *simultaneous* if its timing constraint requires all nodes to halt at the same time. As shown in [69], if T is a simultaneous task, then in any algorithm for solving T, nodes can halt in round r only if there is common knowledge that all nodes are going to halt in round r. For the sake of completeness, we begin with a review of the proof of this property and how it can be applied to derive lower bounds on simultaneous tasks.

Since knowledge formulas are evaluated at timepoints rather than rounds (i.e., between timepoints), it is convenient to associate the act of halting and producing an output value with a time rather than a round. Thus, let us consider only algorithms where the state-space of each node contains a set of special "halting" states, one state for each possible output value, and each node halts and outputs x in round $t+1$ iff the node is in the halting state corresponding to value x at time t. (This is without loss of generality, since any algorithm is easily modified to satisfy this property at the cost of at most one additional round.) Let halt be an atomic proposition that is satisfied in (α, t) iff all nodes of $V(\alpha)$ are in a halting state at time t.

Lemma 4.10 ([69]). *If T is a simultaneous task and $\mathcal{A}$ is an algorithm that solves T, then $(\alpha, t) \models \mathsf{halt}$ iff $(\alpha, t) \models C(\mathsf{halt})$.*

Proof. One direction is immediate: if $(\alpha, t) \models C(\mathsf{halt})$, then by Proposition 4.2, all points in the similarity component of (α, t) satisfy halt, and in particular $(\alpha, t) \models \mathsf{halt}$.

To show the other direction, suppose that $(\alpha, t) \models \mathsf{halt}$. By Proposition 4.2, to show that $(\alpha, t) \models C(\mathsf{halt})$, it is sufficient to show that for all $(\beta, t) \sim (\alpha, t)$ we have $(\beta, t) \models \mathsf{halt}$. This is proven by induction on distance between (α, t) and (β, t) in the similarity graph. The base case, distance zero, is immediate, since we assumed that $(\alpha, t) \models \mathsf{halt}$. For the induction step, suppose that all points at distance k from (α, t) satisfy halt, and let (β, t) be a point at distance $k+1$ from (α, t) in the similarity graph. Then there is some point (γ, t) such that $(\gamma, t) \sim_u (\beta, t)$ for some node $u \in V(\gamma) \cap V(\beta)$, and (γ, t) is at distance k from (α, t). By the induction hypothesis, $(\gamma, t) \models \mathsf{halt}$, and in particular node u is in a halting state at time t in γ. Since $(\gamma, t) \sim_u (\beta, t)$, node u is also in a halting state at time t in β, which means u halts in round $t+1$ of β. But since T is a simultaneous task, *all* nodes must halt in round $t+1$ of β, which implies that all nodes are in halting states at time t and $(\beta, t) \models \mathsf{halt}$. □

The contrapositive of Lemma 4.10 is useful for proving lower bounds on simultaneous actions.

Corollary 4.11. *Suppose that T is a simultaneous task, and φ is a formula such that for any algorithm that solves T, in any point (α, t) of the execution of the algorithm we have $(\alpha, t) \models (\mathsf{halt} \rightarrow \varphi)$. (That is, nodes can halt only if φ holds.) If no full-information protocol acquires common knowledge of φ by time t in any execution, then no algorithm solves task T by time t in any execution.*

Proof. Suppose for the sake of contradiction that $\mathcal{A}$ is an algorithm that solves T and terminates at time $t' \leq t$ in some execution α. Consider the similarity graph induced by the executions of $\mathcal{A}$. By Proposition 4.2, since $(\mathsf{halt} \rightarrow \varphi)$ holds at all points in the similarity graph, $(\mathsf{halt} \rightarrow \varphi)$ is also common knowledge at all points: $(\beta, s) \models C(\mathsf{halt} \rightarrow \varphi)$ for all (β, s).

By Lemma 4.10, we have $(\alpha, t') \models C(\mathsf{halt})$. The common knowledge operator C satisfies the axioms of the $\mathsf{S5}$ modal logic [48], and in particular, it satisfies the axiom $C(\psi \rightarrow \chi) \rightarrow (C(\psi) \rightarrow C(\chi))$ for any ψ, χ. Since $(\alpha, t') \models C(\mathsf{halt} \rightarrow \varphi)$ and $(\alpha, t') \models C(\mathsf{halt})$, we obtain $(\alpha, t') \models C(\varphi)$. This means that $\mathcal{A}$ obtains common knowledge of φ by time $t' \leq t$ in α. It is always possible to simulate any algorithm by a full-information protocol [48], and therefore there is a full-information protocol that acquires common knowledge of φ by time t, contracting the assumption of the corollary. □

For example, in simultaneous consensus, nodes cannot output a decision value of x unless they know that some node received x in its input. If we can show that the fact $\varphi =$ "for either $x = 0$ or $x = 1$, all nodes know that some node received input x" does not become common-knowledge until time t, then we obtain a lower bound of t rounds on simultaneous consensus. Note that this lower bound is stronger than a worst-case lower bound: it is an *all-case* lower bound, which shows that simultaneous consensus cannot be solved in t rounds in *any* execution, not just in *some* execution.

4.3.2 Common Knowledge in Dynamic Graphs

We now characterize the time required to obtain common knowledge in dynamic graphs, towards the goal of obtaining a lower bound on simultaneous tasks (see Corollary 4.11 above). First we must exclude certain "trivial" facts from our results.

Definition 4.12 (Non-trivial facts). *A fact φ is said to be a* non-trivial fact about time t *if for any execution α and time t',*

1. *$(\alpha, t') \models \varphi$ iff $(\alpha, t) \models \varphi$, and*
2. *There exists an execution β that is identical to α up to time $t-1$ (inclusive), over at least as many nodes as participate in α, such that $(\beta, t) \not\models \varphi$.*

For example, the fact that the current time is 6 is trivial — that is, it is not a non-trivial fact about any time. However, the fact that some node received input 0 is a non-trivial fact about time 0. The fact that the network contains exactly 6 nodes may or may not be a trivial fact: if nodes know the size of the network *a priori* (for example because the class of executions contains only executions with 6 nodes), then this fact is trivial, otherwise it may not be.

Recall that we represent an execution as a pair (G, I), where G is the dynamic graph and I is the input assignment. In the sequel we assume that the class $\mathcal{C}$ of executions satisfies the following closure condition:

- If $(G, I), (G, I') \in \mathcal{C}$, where $G = (V, E)$ and $G' = (V', E')$, then for all $U \subseteq V \cup V'$ of size at most $\max\{|V|, |V'|\}$, for all 1-interval connected dynamic graphs H over U, and for all input assignments J satisfying

$$\forall u \in U \quad : \quad \begin{cases} J(u) = I(u) & \text{if } u \in V \setminus V', \\ J(u) = I'(u) & \text{if } u \in V' \setminus V, \text{ and} \\ J(u) \in \{I(u), I'(u)\} \text{ if } u \in V \cap V', \end{cases}$$

  we have $(H, J) \in \mathcal{C}$.

- If $(G, I) \in \mathcal{C}$ and H is a 1-interval connected dynamic graph obtained from G by adding or removing a single edge in one round of G, then $(H, I) \in \mathcal{C}$ as well.

The condition implies that we can span between any two executions by making a sequence of local changes, such as changing the input to a single node, removing a single edge, etc. This intuition will be made formal in Lemma 4.18 below. The condition can be viewed as requiring that the set of executions contain no "topological holes".[3]

Roadmap. Our goal in this section is to show that common knowledge of any non-trivial fact about time t cannot be acquired by time $t + n - 2$; that is, $n - 1$ rounds are required to achieve common knowledge of any non-trivial fact. This holds even if the size of the graph is known in advance to be n (that is, the class $\mathcal{C}$ of executions includes only executions where the size of the graph is n). If the size of the graph is *not* known in advance, we can extend the lower bound by an additional round and show that n rounds are required. Formally, we will show the following:

Theorem 4.13. *If φ is a non-trivial fact about time t, then for all executions α over n nodes and times $t' < t + n - 1$ we have $(\alpha, t') \not\models C\varphi$. Moreover, if $t = 0$ and in α the size of the graph is not known in advance to be at most n, then $(\alpha, n - 1) \not\models C\varphi$ as well.*

[3]The condition is not merely for convenience: it is known that when the topological structure of the space of executions is not sufficiently connected, tasks that are in impossible in the general case become possible. For example, asynchronous fault-tolerant consensus, which is known to be impossible in the general case [54], becomes possible under some restrictions on the global input assignment [115]. We refer to [72] for a comprehensive account of the connection between distributed computation and topology.

Recall from Section 4.1 that a fact φ is common knowledge in (α, t) iff φ holds in every point (β, t) in the similarity component of (α, t). To show that a non-trivial fact φ about time t cannot be common knowledge at any time $t' \in [t, \ldots, t+n-2]$, we will first show that the last $n-2$ rounds of any execution are "mutable", in the sense that we can change them arbitrarily while remaining in the same similarity component. For this part of the proof we will first keep the participants and the input assignment fixed, and focus on the dynamic graph itself. We will show that we can change any aspect of the dynamic graph G at times $t, t+1, \ldots, t+n-2$ while still remaining in the similarity component of $(G, t+n-2)$:

Theorem 4.14. *For every full-information protocol, every set V of $|V| = n$ nodes, all dynamic graphs G and H over V, and all times t and $t' \leq t+n-2$, if G and H agree on all rounds preceding time t, then $((G, I), t') \sim ((H, I), t')$ for every input assignment I for the nodes of V.*

Theorem 4.14 will be proven by showing that we can construct an *indistinguishability chain* spanning between $((G, I), t')$ and $((H, I), t')$. Theorem 4.14 allows us to make arbitrary changes to the dynamic graph in the last $n-2$ rounds, as long as we keep the input assignment and set of participants fixed. However, to prove Theorem 4.13 for time $t = 0$, we must also be able to alter the input assignment and the set of participants, because a non-trivial fact about time 0 may depend on these aspects of the execution. Thus, to complete the proof, we will show in Lemma 4.19 that we can span between any two initial configurations while remaining in the same similarity component. Together with Theorem 4.14, this will give us all the ingredients to prove Theorem 4.13.

The technical details. Fix some input assignment I. The input assignment I will be kept constant throughout the proof of Theorem 4.14; we therefore simplify our notation by omitting I, and use $(G, t) \sim (G', t)$ as short-hand notation for "$((G, I), t) \sim ((G', I), t)$".

To establish Theorem 4.14 we must show that we can span between any two dynamic graphs G, H that are identical up to time $t-(n-2)$ by making only "local changes" that some node cannot observe, obtaining an indistinguishability chain between (G, t) and (H, t). We must maintain 1-interval connectivity at every step of the chain.

If we wish to maintain indistinguishability to some node u at time t, we cannot make any changes to time-nodes in $\mathsf{past}(u, t)$. Therefore a key idea in the proof is *hiding*, an operation that allows us, informally speaking, to "remove" time-nodes from $\mathsf{past}(u, t)$: informally, we say that we *hide* (X, t') from (u, t) when we move inside the similarity component of (G, t) to a point $(G', t) \sim (G, t)$ where (X, t') is hidden from (u, t); in (G', t), node u knows nothing about the states of the nodes in X from time t' onwards. Once we have done this, we can add or remove any edges adjacent only to nodes in X in round t', while still remaining in the similarity component of (G, t), because u does not learn of these changes by time t. In particular, hiding (X, t') from (u, t) allows us to remove more edges from the graph, which in turn lets us hide *more* nodes from (u, t), and so on. In this manner we incrementally transform a dynamic graph into another dynamic graph while remaining in the same similarity component.

Definition 4.15 (Hiding). *Given a graph $G = (V, E)$, nodes $u, v \in V$ and times $t \le t'$, we say that (v, t) is* hidden from (u, t') *in graph G if $(v, t) \notin \mathsf{past}(u, t')_G$. For a set $X \subseteq V$, we say that (X, t) is hidden from (u, t') if (v, t) is hidden from (u, t') for every $v \in X$.*

The hiding set $\mathsf{hide}(G, (X, t), (u, t'))$ *is the set of points (G', t') such that*

(a) $(G, t') \sim (G', t')$ and G' is defined over the same nodes as G,

(b) G and G' are identical up to time t (inclusive), and

(c) (X, t) is hidden from (u, t') in G'.

When can we hide a time-node (v, t) from another time-node (u, t')? Clearly, if $t' \ge t+n-1$, then we always have $\mathsf{past}(u, t')_t = V$ (by Lemma 2.21), so we cannot hope to hide any time-t nodes from (u, t'). However, one might hope that if $t' < t + n - 1$ then we would be able to hide *any* time-t node from (u, t'), and the following lemma shows that this is indeed the case. The lemma is stated in a more general form that allows us to hide sets of nodes rather than just single nodes; this will be useful when we wish to add or remove edges (which requires hiding both endpoints of the edge).

Lemma 4.16 (Hiding Lemma). *Fix a point (G, t), where $G = (V, E)$. For any $k \le n - 2$, set X of size $|X| \le n - k - 1$, and node $u \notin X$, we have* $\mathsf{hide}(G, (X, t - k), (u, t)) \neq \emptyset$.

Proof. We begin with an informal overview of the proof, and then give the technical construction.

Proof overview. The proof is by induction on k. The base case, $k = 0$, is immediate, because at time t node u has not heard from any time-t node but itself.

For the induction step, we fix a set $X \subseteq V$ of size $n - (k + 1) - 1 = n - k - 2$ and a node $u \notin X$. To hide $(X, t - (k + 1))$ from (u, t), we must remove all edges from nodes in X to nodes in $\mathsf{past}(u, t)_{t-k}$ in round $t - k$, while preserving 1-interval connectivity. We can think of the induction hypothesis as providing us with a set of $n - k - 1$ "pebbles" that we can use to cover time-nodes at time $t - k$, hiding them from (u, t) and allowing us to changes the edges adjacent to these nodes in round $t - k$. We make our changes in three steps:

1. Fix some node $w \notin X \cup \{u\}$. In order to preserve 1-interval connectivity, before removing any edges, we will first connect all nodes to node w in round $t - k$. We use two pebbles for this task: for each edge $\{w, v\}$ that we need to add, we first cover nodes w, v at time $t - k$ (hiding them from some third node z at time t), then add the edge $\{w, v\}$ and remove the pebbles so that we can use them again for the next edge.

2. The next step is to remove all edges from nodes in X to nodes outside $X \cup \{w\}$ in round $t-k$, using the same approach as above: for each pair (x, y) where $x \in X$ and $y \notin X \cup \{w\}$, we hide $(\{x, y\}, t-k)$ from (u, t), then remove the edge (x, y) from round $t-k$. 1-interval connectivity is preserved because node w is connected to all nodes of the graph, and we never remove any edges adjacent to node w.

3. Finally, having "fixed" round $t-k$ so that the nodes in X only communicate with nodes in $X \cup \{w\}$, we place all $n-k-1$ of our pebbles on the nodes of $X \cup \{w\}$, which hides $(X, t-k-1)$ from (u, t).

Let us now describe the construction of the similarity chain more formally.

Formal proof. We proceed by induction on k. The base case is $k = 0$, for which the claim is trivial: by Proposition 2.12 we have $\mathsf{past}(u, t)_{G,t} = \{u\}$. Thus, if X is a set such that $u \notin X$, then $\mathsf{past}(u, t)_{G,t} \cap X = \emptyset$ and (X, t) is hidden from (u, t). It follows that $(G, t) \in \mathsf{hide}(G, (X, t), (u, t))$.

For the inductive step, assume that the claim holds for k, and consider $k+1 \leq n-2$. Fix a set $X \subseteq V$ of size $n-(k+1)-1 = n-k-2$ and a node $u \notin X$. To hide $(X, t-(k+1))$ from (u, t), we must remove all edges from nodes in X to nodes in $\mathsf{past}(u, t)_{t-k}$ in round $t-k$, while preserving 1-interval connectivity. We can think of the induction hypothesis as providing us with a set of $n-k-1$ "pebbles" that we can use to cover time-nodes at time $t-k$, hiding them from (u, t) and allowing us to make changes to round $t-k$. We proceed as follows:

1. Fix some node $w \notin X \cup \{u\}$ (there exists such a node, because $|X| \leq n-k-2 \leq n-2$), and fix some ordering $V = \{v_1, \ldots, v_n\}$ where $v_n = w$. We construct a chain $(G, t) = (G_0, t) \sim (G_1, t) \sim \ldots \sim (G_{n-1}, t)$, such that each G_i is identical to G up to time $t-k-1$, and in $G_i = (V, E_i)$ we have $\{v_1, \ldots, v_i\} \times \{w\} \subseteq E_i$. The base case ($i = 0$) is immediate. For the inductive step, to construct G_{i+1} from G_i, we choose some node $z \notin \{w, v_{i+1}\}$, and use the induction hypothesis to find some $G'_i \in \mathsf{hide}(G_i, (\{w, v_{i+1}\}, t-k), (z, t))$ (note that $|\{w, v_{i+1}\}| = 2 \leq n-k-1$, because $k+1 \leq n-2$; therefore the induction hypothesis applies here). Then we add edge $\{w, v_{i+1}\}$ to round $t-k$ of G'_i, obtaining G'_{i+1}. Time-node (z, t) cannot distinguish G'_i from G_{i+1}, because $(\{w, v_{i+1}\}, t-k)$ is hidden from (u, t) in G'_i. Therefore $(G_0, t) \sim \ldots \sim (G_i, t) \sim (G'_i, t) \sim_u (G_{i+1}, t)$. Note also that G_{i+1} is 1-interval connected, because we only add edges to the original graph G.

   The final point, (G_{n-1}, t), satisfies

   (a) $(G, t) \sim (G_{n-1}, t)$,

   (b) G and G_{n-1} are identical up to time $t-k-1$, and

   (c) $(V \setminus \{w\}) \times \{w\} \cup \{w\} \times (V \setminus \{w\}) \subseteq E_{n-1}(t-k)$, where $G_{n-1} = (V, E_{n-1})$.

2. Next we remove all edges from nodes in X to nodes outside $X \cup \{w\}$ in round $t-k$, by constructing a chain exactly as above: for each pair (x, y) where $x \in X$ and $y \notin X \cup \{w\}$, we hide $(\{x, y\}, t-k)$ from (u, t), then remove the edge (x, y) from round $t-k$. At every step, 1-interval connectivity is preserved, because node w has bidirectional edges to all nodes of the graph, and we never remove any edges adjacent to node w. Eventually we obtain a point (H, t) such that
   (a) $(H, t) \sim (G_{n-1}, t)$,
   (b) H and G_{n-1} are identical up to time $t-k-1$, and
   (c) $E_H(t-k) \cap (X \times (V \setminus (X \cup \{w\}))) = \emptyset$, where $H = (V, E_H)$. That is, all edges from nodes in X in round $t-k$ hit nodes in $X \cup \{w\}$.
3. Since $|X \cup \{w\}| = |X| + 1 \leq n-k-1$, the induction hypothesis implies that $\mathsf{hide}(H, (X \cup \{w\}, t-k), (u, t)) \neq \emptyset$. Fix some $(H', t) \in \mathsf{hide}(H, (X \cup \{w\}, t-k), (u, t))$. We have
   (a) $(H', t) \sim (H, t)$,
   (b) H' and H are identical until time $t-k$, and
   (c) $\mathsf{past}(u, t)_{H', t-k} \cap (X \cup \{w\}) = \emptyset$.

   We claim that $(H', t) \in \mathsf{hide}(G, (X, t-k-1), (u, t))$. This holds because
   (a) $(H', t) \sim (H, t) \sim (G_{n-1}, t) \sim (G, t)$, and therefore $(H', t) \sim (G, t)$;
   (b) H' is identical to H until time $t-k$, and combining the properties of H and G_{n-1}, we obtain that H' is identical to G until time $t-k-1$; and finally,
   (c) $\mathsf{past}(u, t)_{H', t-k-1} \cap X = \emptyset$: fix $x \in X$. By construction of H and the fact that H and H' are identical in round $t-k$, all edges from x in round $t-k$ hit nodes in $X \cup \{w\}$. However, because $(X \cup \{w\}, t-k)$ is hidden from (u, t) in H', we have $(y, t-k) \not\rightsquigarrow (u, t)$ for every $y \in X \cup \{w\}$. It follows that $(x, t-k-1) \not\rightsquigarrow (u, t)$.

□

We can use the Hiding Lemma to prove Theorem 4.14:

Proof of Theorem 4.14. Let G, H be graphs over the same nodes that are identical up to time t, and let $t' \leq t+n-2$. Graphs G, H can differ only in rounds $t+1, \ldots, t'$ (or after time t'). Lemma 4.16 shows that for every $u, v, w \in V$, graph G' and time $t+1 \leq r \leq t'$, the set $\mathsf{hide}(G, (\{v, w\}, t'), (u, t))$ is non-empty, that is, we can hide $(\{v, w\}, r)$ from (u, t') without altering any round preceding time t. (We instantiate Lemma 4.16 with $k = t' - r$, $|X| = 2$, which is permissible because the lemma allows any set X of size at most $n-k-1 = n-(t'-r)-1 \geq n-(t+n-2)+(t+1)-1 = 2$.)

We can span between (G, t) and (H, t) as follows: we iterate over rounds $r = t+1, \ldots, t'$. For each round r,

1. For each edge $(x, y) \in G(r) \setminus H(r)$, hide the endpoint $\{x, y\}$ in round r from some node $w \notin \{x, y\}$ (formally, we use Lemma 4.16 to move to a point in $\mathsf{hide}(G, (\{x, y\}, r), (w, t')))$. Then add the edge $\{x, y\}$ to round r. Node w cannot observe the difference at time t'.

   At the end of this stage, we have a similarity chain $(G, t') \sim (F, t')$, where F is a dynamic graph obtained by taking the union of the edges of $G(r)$ and $H(r)$ in each round r.

2. Next, for each edge $(x, y) \in H(r) \setminus G(r)$, hide the endpoint $\{x, y\}$ in round r from some node $w \notin \{x, y\}$ as before, and *remove* $\{x, y\}$ from round r. Note that 1-interval connectivity is preserved, because we do not remove any edges of H, and H is 1-interval connected.

   At the end of this stage we have a similarity chain $(F, t') \sim (H, t')$.

This shows that $(G, t') \sim (H, t')$. □

Let us now turn our attention to Theorem 4.13. We have already shown (Theorem 4.14) that we can transform the last $n-2$ rounds of a dynamic graph arbitrarily. Now, to deal with time 0, we must also show that we can span between any two initial configurations. Specifically we will show that we can span between any two executions over the same number n of nodes, maintaining similarity at least until time $n-2$, and furthermore, we will show that if the execution space admits any execution over more than n nodes, then we can maintain similarity up to time $n-1$.

Theorem 4.14 forms a crucial part of the argument: when we wish to change the input to some node u or replace node u with another node, Theorem 4.14 allows us to re-arrange the graph into a static line graph with u at one end of the line. We can then swap out node u and/or change its input, and since there are n nodes in the line, the node at the other end of the line cannot notice this change by time $n-2$. The following lemma captures this single step:

Lemma 4.17. *Fix a set U of $|U| = n$ nodes, a node v (which may or may not be in U), and let $V := (U \setminus \{u\}) \cup \{v\}$. Let I be some input assignment to the nodes of U, and let J be an input assignment to V such that $I(x) = J(x)$ for all $x \notin \{u, v\}$. Finally, let $\alpha = (G, I)$ be some execution over the nodes in U, and let $\beta = (H, J)$ be some execution over the nodes in V. Then $(\alpha, n-2) \sim (\beta, n-2)$.*

Proof. Fix an order $U = \{u_0, \ldots, u_{n-1}\}$ where $u_0 = u$. Then we can write $V = \{v, u_1, \ldots, u_{n-1}\}$. Let L be a static line graph $u, u_1, \ldots, u_{n-1}$, and let L' be the static line $v, u_1, \ldots, u_{n-1}$. Since L and L' agree on $\mathsf{past}(u_{n-1}, n-2)$ (the only change occurs at distance $n-1$ from u_{n-1}), and input assignments I and J do as well, we have $((L, I), n-2) \sim_{u_{n-1}} ((L', J), n-2)$. By Theorem 4.14 we have $(\alpha, n-2) \sim ((L, I), n-2) \sim_{u_{n-1}} ((L', J), n-2) \sim (\beta, n-2)$, and the claim follows. □

By repeatedly applying Lemma 4.17 we can span between any two executions, preserving similarity until time $n-2$, where n is the size of the smaller of the two executions.

Lemma 4.18. *For any two executions α, β over at least n nodes we have $(\alpha, n-2) \sim (\beta, n-2)$.*

Proof. We can span between α and β using a sequence of intermediate executions $\alpha_0 = \alpha, \alpha_1, \ldots, \alpha_m = \beta$, where each α_{i+1} is obtained from α_i by replacing a single node u participating in α with a node v participating in β, and also changing the input assignment accordingly. By Lemma 4.17 (together with Proposition 4.1) we have $(\alpha_i, n-2) \sim (\alpha_{i+1}, n-2)$ for every $i = 0, \ldots, m-1$, and the claim follows. $\square$

Lemma 4.18 and Theorem 4.14 yield the first part of Theorem 4.13: together these claims imply (informally) that no aspect of the execution that depends on time $t \geq 0$ can become common knowledge by time $t + n - 2$. To show the second part of Theorem 4.13, we will show that we can if $\mathcal{C}$ contains some execution over at least $n+1$ nodes, we can extend similarity by one extra round and obtain a similarity chain at time $n-1$.

Lemma 4.19. *If $\mathcal{C}$ contains an execution over more than n nodes, then for any two executions α, β over at least n nodes we have $(\alpha, n-1) \sim (\beta, n-1)$.*

Proof. Suppose that there is some execution $\gamma \in \mathcal{C}$ with at least $n+1$ nodes. If α and β both contain at least $n+1$ nodes, then Lemma 4.18 yields $(\alpha, n-1) \sim (\beta, n-1)$, and we are done. Thus, assume (without loss of generality) that α contains exactly n nodes.

Let $U = \{u_1, \ldots, u_n\}$ be the nodes participating in α, and let $V = \{v_1, \ldots, v_k\}$ be the nodes of β (where $k \geq n$). From the existence of γ, we know that there is some node $w \notin U$ that does not participate in α, and from the condition on the execution space $\mathcal{C}$, any execution obtained from an execution in $\mathcal{C}$ by adding node w with the input it has in γ (in a manner that preserves 1-interval connectivity) is an execution of $\mathcal{C}$.

By Lemma 4.16, there exists an execution $\alpha_1 \in \mathsf{hide}(\alpha, (u_1, 1), (u_2, n-1))$.[4] Execution α_1 satisfies $(u_1, 1) \notin \mathsf{past}(u_2, n-1)_{\alpha_1}$ and $(\alpha_1, n-1) \sim (\alpha, n-1)$. Let α_2 be the execution obtained from α_1 by adding node w with its input in γ, and adding the edge $\{w, u_1\}$ to each round. Since $(u_1, 1) \notin \mathsf{past}(u_1, n-1)_{\alpha_1}$, there is no time $t \in \{1, \ldots, n-1\}$ such that $(u_1, t) \in \mathsf{past}(u_1, n-1)_{\alpha_1}$; that is, $\mathsf{past}(u_2, 1)_{\alpha_1} \subseteq (\{u_2, \ldots, u_n\} \times \{0, \ldots, n-1\}) \cup \{(u_1, 0)\}$. By definition, α_1 and α_2 are $(\{u_2, \ldots, u_n\}, n-1)$-identical, and also $(u_1, 0)$-identical (because node u_1 receives the same input in both executions). Therefore Proposition 2.28 shows that $(\alpha_1, n-1) \sim_{u_2} (\alpha_2, n-1)$. Together with the fact that $(\alpha_1, n-1) \sim (\alpha, n-1)$, we obtain $(\alpha, n-1) \sim (\alpha_2, n-1)$.

If $k \geq n+1$, that is, β already contains at least $n+1$ nodes, then we define $\beta_2 := \beta$. Otherwise we know that there exists a node that does not participate in β, and we repeat the process above to obtain an execution β_2 over at least $n+1$ nodes such that $(\beta, n-1) \sim (\beta_2, n-1)$. Since α_2 and β_2 both contain at least $n+1$ nodes, the first part of the theorem shows that $(\alpha_2, n-1) \sim (\beta_2, n-1)$; therefore we obtain $(\alpha, n-1) \sim (\alpha_2, n-1) \sim (\beta_2, n-1) \sim (\beta, n-1)$, and we are done. $\square$

[4]More precisely, there exists a dynamic graph G_{α_1} such that $(G_{\alpha_1}, n-1) \in \mathsf{hide}(G, (u_1, 1), (u_2, n-1))$, and we define $\alpha_1 := (G_{\alpha_1}, I)$.

Now we can put all the ingredients together to prove Theorem 4.13.

Proof of Theorem 4.13. Let φ be a non-trivial fact about time t, and let $t' < t+n-1$, that is, $t' \leq t+n-2$. Fix an execution α such that $(\alpha, t) \models \varphi$. (Note that common knowledge of φ cannot be held at time t unless φ holds at time t.) We divide into two cases:

1. $t > 0$. Then exists an execution β which is identical to α up to time $t-1 \geq 0$, such that $(\beta, t) \not\models \varphi$. Let G_α, G_β be the dynamic graphs in α, β respectively. Since α, β are identical up to time 0 (at least), G_α and G_β are over the same nodes, and all nodes receive the same inputs in α and β. By Theorem 4.14 (together with Proposition 4.1) we have $(G_\alpha, t') \sim (G_\beta, t')$, as α, β are identical up to time $t-1 \geq t'-(n-2)$, and since the inputs are the same in both executions we also have $(\alpha, t') \sim (\beta, t')$.

   Because φ is a fact about time t, and $(\beta, t) \not\models \varphi$, we also have $(\beta, t') \not\models \varphi$. It follows that $(\alpha, t') \not\models C\varphi$.

2. $t = 0$. Then $t' < t+n-1$, and there exists an execution β, over at least n nodes, such that $(\beta, 0) \not\models \varphi$ and thus also $(\beta, t') \not\models \varphi$. Using Lemma 4.18 and Proposition 4.1 we have $(\alpha, t') \sim (\beta, t')$, and therefore $(\alpha, t') \not\models C\varphi$.

Now suppose that the number of nodes participating in α is not known in advance to be at most n. This implies that there exists some execution γ with more than n nodes. Since φ is a non-trivial fact about time t, there exists an execution β over at least n nodes, such that $(\beta, t) \not\models \varphi$, and therefore also $(\beta, n-1) \not\models \varphi$. By Lemma 4.19 we have $(\alpha, n-1) \sim (\beta, n-1)$, so $(\alpha, n-1) \not\models C\varphi$. □

4.3.3 Application to Simultaneous Consensus

The characterization of common knowledge from Theorem 4.13 yields the following lower bound on simultaneous consensus:

Theorem 4.20. *For any class $\mathcal{C}$ of executions,*

(a) Even if all executions in $\mathcal{C}$ are over a fixed set of n nodes, simultaneous consensus requires at least $n-1$ rounds in every execution of $\mathcal{C}$.

(b) Moreover, if $\mathcal{C}$ contains executions over more than n nodes (that is, if the size of the graph is not known in advance to be at most n), then in every n-node execution in $\mathcal{C}$, simultaneous consensus requires at least n rounds.

Proof. Let us first show part (a). Assume that the set V of participants is the same in all executions in $\mathcal{C}$.

Let φ_x stand for the proposition "some node has input x". Then the formula $(E\varphi_0 \vee E\varphi_1)$ is a non-trivial formula about time 0: clearly its truth value depends only on the inputs (i.e., only on the state at time 0). We also claim that no execution β satisfies $(\beta, 0) \models (E\varphi_0 \vee E\varphi_1)$, so this fact is non-trivial.

Claim. *No execution $\beta \in \mathcal{C}$ satisfies $(\beta, 0) \models (E\varphi_0 \vee E\varphi_1)$.*

Proof. Any node whose input is 0 cannot distinguish $(\beta, 0)$ from an execution where all nodes receive input 0, so if there exists a node u with input 0 we have $(\beta, 0) \not\models K_u\varphi_1$ and therefore $(\beta, 0) \not\models E\varphi_1$. Similarly, if there is a node with input 1, then $(\beta, 0) \not\models E\varphi_0$. Thus, if in β some node receives input 0 and some node receives input 1, then $(\beta, 0) \not\models (E\varphi_0 \vee E\varphi_1)$.

Now suppose that in β all nodes receive input 0 (the case where all inputs are 1 is similar). There is an execution β' whose initial configuration is the same as in β, except that some node u receives input 1. Any node $v \neq u$ cannot distinguish $(\beta, 0)$ from $(\beta', 0)$, and therefore $(\beta, 0) \not\models E\varphi_0$. But also $(\beta, 0) \not\models E\varphi_1$, because this would imply that $(\beta, 0) \models \varphi_1$ (which is false). So again we have $(\beta, 0) \not\models (E\varphi_0 \vee E\varphi_1)$. □

To show that $(E\varphi_0 \vee E\varphi_1)$ is a non-trivial fact about time $t = 0$, we must show that for any execution α and time t', there exists an execution β that is identical to α until time $t - 1 = -1$, over at least as many nodes as α, such that $(\beta, 0) \not\models (E\varphi_0 \vee E\varphi_1)$. By the claim we showed above, we can simply set $\beta := \alpha$: we have $(\alpha, 0) \not\models (E\varphi_0 \vee E\varphi_1)$, and obviously α is identical to itself. Therefore $(E\varphi_0 \vee E\varphi_1)$ is a non-trivial fact about time 0. By Theorem 4.13, $(E\varphi_0 \vee E\varphi_1)$ does not become common knowledge until time $n - 1$ in any execution.

To apply Corollary 4.11, we must also show that for any simultaneous consensus algorithm, all points (α, t) satisfy $(\alpha, t) \models \mathsf{halt} \rightarrow (E\varphi_0 \vee E\varphi_1)$: suppose for the sake of contradiction that there is a point (α, t) such that $(\alpha, t) \models \mathsf{halt}$ but $(\alpha, t) \not\models E\varphi_0 \vee E\varphi_1$. Let x be the decision value in (α, t) (that is, the value output by all nodes in round $t + 1$ of α). Since $(\alpha, t) \not\models E\varphi_0 \vee E\varphi_1$, we also have $(\alpha, t) \not\models E\varphi_x$, so there is a point (β, t) and node $u \in V(\alpha)$ such that $(\alpha, t) \sim_u (\beta, t)$ but $(\beta, t) \not\models \varphi_x$. Since $(\alpha, t) \sim_u (\beta, t)$ and node u is "locked in" to a decision value of x in (α, t), node u also outputs x in execution β, where no node has input x (because φ_x is not satisfied), contradicting validity.

Now we can apply Corollary 4.11, which yields a lower bound of $n - 1$ rounds on simultaneous consensus in any execution. This completes the proof or part (a). To obtain part (b), note that if the number of participants is *not* known in advance, we obtain from Theorem 4.13 a lower bound of n rounds (instead of $n - 1$ rounds) on acquiring common knowledge of $(E\varphi_0 \vee E\varphi_1)$ in any execution. Corollary 4.11 then yields a lower bound of n rounds on simultaneous consensus. □

Corollary 4.21. *When the count is not known in advance, obtaining an upper bound on the count* $(0, 0)$*-reduces to simultaneous consensus.*

Proof. Simply run the simultaneous consensus algorithm, and when it halts, output the current time. □

To illustrate the stark difference between simultaneous consensus and "plain" consensus (with no timing constraints), let us show there is a class of executions where plain consensus can decide in $O(1)$ rounds, while simultaneous consensus still requires $n - 1$ rounds (by Theorem 4.20). Consider the following problem:

Definition 4.22 (Clique detection). *In the* clique detection *problem, each node must eventually output "yes" or "no", such that*

(a) If the communication graph in each round is a clique, all nodes output "yes"; and

(b) If any node outputs "yes", then the communication graph in the first round is a clique.

If the communication graph in the first round is a clique but in subsequent rounds it is not a clique, nodes may output "yes" or "no" arbitrarily.

Theorem 4.23. *There is a clique detection algorithm that terminates in 4 rounds and uses messages of size $O(\log n)$.*

Before we give the clique detection algorithm, note its application to consensus:

Corollary 4.24. *There is a consensus algorithm using messages of size $O(\log n)$ such that in some executions, all nodes decide in 4 rounds.*

Proof. To solve consensus, we can execute the clique detection algorithm side-by-side with a more "pessimistic" consensus algorithm, e.g., the token-dissemination algorithm from Chapter 3. In the first round nodes also append their input value to the message they send. If the clique detection algorithm outputs "yes" in round 4, we decide on the minimum input received in the first round; otherwise we continue to execute the pessimistic consensus algorithm until it decides. □

In contrast, Theorem 4.20 shows that simultaneous consensus can *never* be reached in less than $n-1$ rounds, even when messages are not bounded in size and the graph is a static clique.

Let us now prove that an $O(1)$-round clique detection algorithm exists.

Proof of Theorem 4.23. Consider the following algorithm.

- In round 1, all nodes send their UIDs.
- In round 2, each node examines the set of the UIDs it received in round 1, and treats them as an ordered cycle. It sends its own UID plus the next UID after its own in cyclic order; i.e., if its UID is not the largest it sends the next largest UID, and if its UID is the largest it sends the smallest UID.
- In round 3, each node checks if the messages it received in round 2 are consistent with the set of UIDs it received in round 1: that is, if X is the set of UIDs received in round 1 (including the node's own UID), then the node expects to receive exactly the pairs (u, v) that form the ordered Hamiltonian cycle over X. If this holds, the node sends its UID + "yes", otherwise it sends its UID + "no".
- If only "yes" votes were received in round 3 (including the node's own vote), and all nodes heard in round 1 were received again in round 3, output "yes"; otherwise output "no".

It is easy to see that if the graph is always a clique then all nodes output "yes". The other direction is more interesting: assume that the graph in the first round is *not* a clique, i.e., some edge (x, y) is missing from the graph in round 1. We will show that each node either votes "no" in round 3 or receives a "no" vote in this round, and hence all nodes output "no" in round 4.

Let X_v denote the set of UIDs received by node $v \in V$ in round 1, that is, the set of v's in-neighbors in round 1, including node v itself. Let u be a node that outputs "yes" at the end of the algorithm. Then

($\star$) In round 2 node u receives the directed cycle over X_u, and in particular, all nodes in X_u send UIDs of other nodes in X_u in round 2; and

($\star\star$) In round 3 node u receives "yes" votes from all the nodes in X_u.

We will now show that

I. $X_u = V$, that is, node u hears from everyone in the first round.

II. Using this fact together with ($\star$) and ($\star\star$), we will show that $E(1) = V^2$, that is, the graph is a clique in the first round.

I. Node u hears from everyone in the first round: suppose for the sake of contradiction that $X_u \neq V$. From strong connectivity, there is some edge (z, w) in the directed cut $(V \setminus X_u, X_u)$ in round 1. We have $w \in X_u$ and $z \in X_w \setminus X_u$.

By ($\star\star$), since $w \in X_u$, we know that w votes "yes" in round 3. Therefore, the pairs received by node w in round 2 form the ordered cycle $x_0 \rightarrow x_1 \rightarrow \ldots \rightarrow x_{k-1} \rightarrow x_0$ over X_w, where $x_0 = w$ and $k = |X_w|$. Since $z \in X_w$, there is some index i such that $x_i = z$. We know from ($\star$) that all nodes in X_u send the UIDs of other nodes in X_u in round 2; since $x_{(i-1) \bmod k}$ sends $x_i = z \notin X_u$ in round 2, it must be that $x_{(i-1) \bmod k} \notin X_u$ as well. By the same logic we must also have $x_{(i-2) \bmod k} \notin X_u$, and continuing backwards along the cycle in this manner we can show that $x_0 \notin X_u$. But this is a contradiction, because $x_0 = w \in X_u$ by choice of w. It follows that $X_u = V$.

II. The graph is a clique in the first round: suppose not, that is, assume that some edge (x, y) is missing from $E(1)$. We have already shown in (I) that $X_u = V$, and in particular $y \in X_u$. Therefore, by ($\star\star$), node y votes "yes" in round 3. This implies that in round 2 node y receives an ordered cycle C_y of the nodes in X_y. Because we assumed that $(x, y) \notin E(1)$, we have $x \notin X_y$, so x does not appear in C_y.

We also know by ($\star$) that node u receives in round 2 the complete ordered cycle C_u over V, since ($\star$) asserts that node u receives the ordered cycle over X_u, and we showed in (I) that $X_u = V$. In particular, C_u does include node x. Note that C_y and C_u are both subgraphs of the graph $H = (V, F)$, where

$$F := \{(z, w) \mid \text{ node } z \text{ sends the pair } (z, w) \text{ in round } 2\}.$$

The out-degree of each node in H is exactly 1, and C_u is a cycle over all nodes of V; therefore C_y must equal C_u. But C_u includes x while C_y does not, and this is a contradiction. $\square$

4.4 Δ-Coordinated Consensus

Since simultaneous consensus is expensive and requires $n-1$ rounds even in very well-behaved executions, it is interesting to consider a trade-off between the performance of the consensus algorithm and the degree of coordination it achieves. To this end we consider the following problem:

Definition 4.25 (Δ-Coordinated Consensus). *A protocol solves* Δ-coordinated consensus *if it solves consensus, and in addition, all nodes output their decision values within a window of Δ rounds. This timing constraint is called* Δ-coordination.

In the sequel, to simplify the presentation, we assume that the set V of participants is fixed (although our algorithms require only an upper bound on the count).

One might expect that Δ-coordinated consensus would not be much easier than simultaneous consensus. For example, when $\Delta = 1$, we require all nodes to decide within one round of each other; it seems that if we can achieve this, then simultaneous coordination can be achieved at not much extra cost — a cost of Δ additional rounds, perhaps. Indeed, in the worst case this expectation is borne out by the following lower bound.

Theorem 4.26. *For any Δ-coordinated consensus algorithm, there exists an execution in which no node decides before round $n - \Delta - 1$, even when n is known* a priori.

Proof. Suppose that there exists a Δ-coordinated consensus algorithm $\mathcal{A}$, such that in every execution some node decides before time $R < n - \Delta - 1$. Then in $\mathcal{A}$, *all* nodes decide no later than time $R + \Delta < n - 1$ in every execution. We can obtain an algorithm for simultaneous consensus in fewer than $n-1$ rounds by simply having each node run $\mathcal{A}$ and output $\mathcal{A}$'s decision value at time $R + \Delta < n-1$, contradicting the lower bound from Section 4.3. □

This result shows the existence of only one "bad" execution where no node can decide until time $n - \Delta - 1$. Given the general similarity between Δ-coordinated consensus and simultaneous consensus, one might expect that a Δ-coordinated consensus protocol would *never* be able to decide before time $n - \Delta - 1$ (just as simultaneous consensus can never decide before time $n - 1$). However, we now show that even in 1-coordinated consensus, nodes can sometimes decide significantly earlier than time $n - \Delta - 1$. Consider the following protocol.

Clear-Majority Protocol Fix some integer $k_{\max}$, and for each $k = 1, \ldots, k_{\max}$, let $t_k := n - k \cdot \Delta - 1$. In each round the nodes forward the set of all node UIDs they have heard from so far, along with the input to each node. At time t_k, an undecided node decides v iff it has heard of at least $\lfloor n/2 \rfloor + 1 + \binom{k}{2}\Delta$ inputs equal to v. Finally, at time $n-1$, all the nodes know all the inputs; at this point any undecided node decides on the majority input (breaking ties in some consistent way if there is no majority).

Lemma 4.27. *The clear-majority protocol solves Δ-coordinated consensus. Furthermore, when the fraction of identical inputs is at least $(1/2+\epsilon)n$ for some constant ϵ, and if $\Delta \le (\epsilon n - 1)/2$, all nodes can decide after $n - \Theta(\sqrt{n\Delta})$ rounds.*

Proof. Agreement and validity follow immediately from the fact that nodes always decide on the majority value (or, if there is no majority value, all nodes reach time $n-1$ and decide in some consistent way). To show that the protocol is Δ-coordinated, suppose that in some execution, the earliest node u decides on value v at time t_k. We must show that all nodes decide no later than time $t_k + \Delta = t_{k-1}$.

Because the communication graph in every round is connected, for all $s \le n-1$, at time $n-s-1$ in the execution each node has heard all but at most s of the inputs. In particular, by time $t_{k-1} = n-(k-1)\Delta - 1$ each node has heard all but $(k-1)\Delta$ of the inputs. Since u decides v at time t_k, the input assignment contains at least $\lfloor n/2 \rfloor + 1 + \binom{k}{2}\Delta$ values equal to v, and hence by time t_{k-1} each node hears at least $\lfloor n/2 \rfloor + 1 + \binom{k}{2}\Delta - (k-1)\Delta = \lfloor n/2 \rfloor + 1 + \binom{k-1}{2}\Delta$ inputs equal to v. Thus, all nodes that do not decide at time t_k decide v at time $t_{k-1} = t_k + \Delta$, as required.

Now suppose that for some constant ϵ, the input assignment contains at least $(1/2+\epsilon)n$ copies of some value v. By time $t_k = n - k\cdot\Delta - 1$ each node hears all but $k \cdot \Delta$ of the input values, i.e., at least $(1/2+\epsilon)n - k\cdot\Delta$ copies of v. If $\Delta \le (2\epsilon n - 1)/4$, we set $k_{\max} = \lfloor \sqrt{(2\epsilon n - 1)/\Delta} - 1 \rfloor$, and then simple algebra shows that $(1/2+\epsilon)n - k_{\max}\cdot\Delta \ge \binom{k_{\max}}{2} + \lfloor n/2 \rfloor + 1$; thus, by time $t_{k_{\max}}$, each node hears sufficiently many copies of v to decide. For this value of $k_{\max}$ we have $t_{k_{\max}} = n - \Theta(\sqrt{n\Delta})$. □

The clear-majority protocol can be viewed as an instance of a more general scheme, in which nodes decide as soon as they know that everyone else will decide the same value within Δ rounds. We now present this scheme and use it to derive another Δ-coordinated consensus algorithm, one that performs well when the dynamic diameter is small.

A general transformation for achieving Δ-coordination. The timing constraint in Δ-coordinated consensus allows nodes to halt only if they *know* that all other nodes will also have halted within Δ rounds. To make this intuition formal, let us associate halting with a time rather than a round (as in Section 4.3.1), and let $\mathsf{halt}_{u,t}$ be an atomic proposition that holds true iff node u is in a halting state at time t in the execution. The Δ-coordination constraint now requires nodes to enter halting states at times that are separated by no more than Δ rounds. We assume that once a node enters a halting state it remains in it forever, and say that a node *halts* when it first enters a halting state.

Proposition 4.28. *An algorithm satisfies Δ-coordination iff all points (α, t) in its execution satisfy*

$$\forall u \in V \quad : \quad (\alpha, t) \models \left(\mathsf{halt}_{u,t} \rightarrow K_u\left(\bigwedge_{v\in V} \mathsf{halt}_{v,t+\Delta}\right)\right). \tag{4.4.1}$$

Proof. The first direction is easy: suppose that an algorithm satisfies (4.4.1), and node u halts in (α, t). By (4.4.1) we have $(\alpha, t) \models K_u \left(\bigwedge_{v\in V} \mathsf{halt}_{v,t+\Delta}\right)$, and since a node cannot know a fact unless it is true, $(\alpha, t) \models \bigwedge_{v\in V} \mathsf{halt}_{v,t+\Delta}$. Therefore all nodes halt no later than time $t + \Delta$, as required.

To show the other direction, fix a Δ-coordinated algorithm, and suppose for the sake of contradiction that for some point (α, t) and node u we have $(\alpha, t) \models \mathsf{halt}_{u,t}$ but $(\alpha, t) \not\models K_u \left(\bigwedge_{v\in V} \mathsf{halt}_{v,t+\Delta}\right)$. Then there is an execution β such that $(\alpha, t) \sim_u (\beta, t)$ but $(\beta, t) \not\models \bigwedge_{v\in V} \mathsf{halt}_{v,t+\Delta}$. This implies that there is some node $v \in V$ such that $(\beta, t) \not\models \mathsf{halt}_{v,t+\Delta}$, that is, node v is not in a halting state at time $t + \Delta$ in β. Since $(\alpha, t) \sim_u (\beta, t)$ and $(\alpha, t) \models \mathsf{halt}_{u,t}$, node u is also in a halting state in (β, t), which means it halts more than Δ rounds before node v. Therefore the algorithm does not satisfy Δ-coordination. □

Now suppose that we are given an (untimed) consensus algorithm $\mathcal{A}$, and we wish to transform it into a Δ-coordinated consensus algorithm. Since $\mathcal{A}$ is a consensus algorithm, it already ensures agreement and validity; all we need to do is delay the outputs so as to satisfy (4.4.1). One way to do so is to explicitly use the condition from (4.4.1),

$$K_u \left(\bigwedge_{v\in V} \mathsf{halt}_{v,t+\Delta} \right), \tag{4.4.2}$$

as the condition that determines whether node u halts at time t. However, there is some circularity inherent in this description: the termination condition in (4.4.2) determines at what points the formulas $\mathsf{halt}_{v,t}$ are satisfied (as it determines when nodes halt), but this in turn determines whether (4.4.2) holds or not. Any algorithm that abides by the termination condition in (4.4.2) is in fact a *fixpoint* of this circular process. Next we describe how an (untimed) consensus algorithm can be transformed into a conservative approximation of such a fixpoint, satisfying (4.4.1) and guaranteeing Δ-coordination.

Fix a full-information consensus algorithm $\mathcal{A}$ which always decides by time $n - 1$, and let "$\mathsf{val}_{\mathcal{A}}(v)$" stand for the formula that asserts that (α, t) is v-valent with respect to $\mathcal{A}$; that is, in any possible extension of the first t rounds of α, all nodes decide v under $\mathcal{A}$. Let $K_u^{@t}\varphi$ be short-hand notation for the formula that means "node u knows that at time t fact φ will hold", and let $E^{@t}\varphi := \bigwedge_{u\in V} K_u^{@t}\varphi$. Also, since we now study protocols where nodes reason over the executions of other protocols, let us be more explicit about the protocol with respect to which we evaluate knowledge formula: we use $\mathcal{R}(\mathcal{A})$ to denote the similarity graph induced by all the executions of algorithm $\mathcal{A}$, and let $(\mathcal{R}(\mathcal{A}), \alpha, t)$ denote time t in the run of algorithm $\mathcal{A}$ in execution $\alpha = (G, I)$, where G is a dynamic graph and I is the input assignment.

Now we can state the transformation:

The Δ-Ladder. Given an eventual consensus algorithm $\mathcal{A}$ in which all nodes decide no later than time $n - 1$, we first transform $\mathcal{A}$ into an equivalent full-information algorithm $\mathcal{A}'$. In $\mathcal{A}'$, nodes store their entire history in their local state and send it every message. Each node locally computes from its state (i.e., its history) the corresponding state that it would be in under the original algorithm $\mathcal{A}$, and uses the computed state to determine when to halt and what value to output in accordance with $\mathcal{A}$.

Next we construct a Δ-coordinated algorithm $\mathcal{B} = \mathcal{B}(\mathcal{A}')$ that uses $\mathcal{A}'$ to obtain a consensus value but uses its own rules to decide when to halt. In $\mathcal{B}$, nodes emulate $\mathcal{A}'$, but do not immediately output its decisions. Instead, each undecided node u evaluates the following decision rules at each decision point $t_k = n - \Delta \cdot k - 1$. The rules are defined recursively in k, and accordingly, we list them in reverse order of decision time:

- Decide v at time $n-1$ if $(\mathcal{R}(\mathcal{A}'), \alpha, n-1) \models K_u\left(\mathsf{val}_{\mathcal{A}'}(v)\right)$, that is, if it is known that the run is v-valent for $\mathcal{A}'$. Here α is the execution of $\mathcal{A}'$ that is being emulated by the nodes.

  This rule is a catch-all "default case" which simply ensures that all nodes halt no later than time $n-1$. Note that since $\mathcal{A}'$ decides no later than time $n-1$, the rule is always satisfied at time $n-1$ for either $v = 0$ or $v = 1$.

- Decide v at time $n - \Delta - 1$ if

  $$(\mathcal{R}(\mathcal{A}'), \alpha, n - \Delta - 1) \models K_u E^{@(n-1)} \mathsf{val}_{\mathcal{A}'}(v).$$

  That is, each node decides v at time $n - \Delta - 1$ if it knows that the rule for deciding v at time $n-1$ (listed above) will be satisfied for all nodes.

- Decide v at time $n - 2\Delta - 1$ if

  $$(\mathcal{R}(\mathcal{A}'), \alpha, n - 2\Delta - 1) \models K_u E^{@(n-\Delta-1)} E^{@(n-1)} \mathsf{val}_{\mathcal{A}'}(v).$$

  That is, each node decides v at time $n - 2\Delta - 1$ if it knows that the rule for deciding v at time $n - \Delta - 1$ (listed above) will hold for everyone.

- And so on.

In general, the decision rules are a set of knowledge formulas $\varphi_0(v), \ldots, \varphi_{k_{\max}}(v)$ for $v \in \{0, 1\}$, defined recursively by $\varphi_0 := \mathsf{val}_{\mathcal{A}'}(v)$ and $\varphi_{i+1} := E^{@(n-i\cdot\Delta-1)}\varphi_i$. An undecided node u decides at time $n - k \cdot \Delta - 1$ iff $K_u\varphi_k$ holds.

We remark that since $\mathcal{A}'$ is a full-information algorithm, the messages it sends allow nodes to eventually construct a complete picture of the dynamic graph and the messages received by the other nodes (with some delay, depending on the dynamic diameter of the graph). Thus, nodes are able to reason not only about their own knowledge, which is determined solely by their local state, but also about the knowledge of other nodes. To evaluate the rules above, nodes use the information they have about the dynamic graph to compute the local states of the other nodes and thereby determine what the other nodes know.

It is easy to see that any instantiation of this scheme satisfies Δ-coordinated consensus.

Lemma 4.29. *Let $\mathcal{A}$ be a consensus algorithm and $\mathcal{A}'$ be its full-information version. Then $\mathcal{B} = \mathcal{B}(\mathcal{A}')$ solves Δ-coordinated consensus.*

Proof. Agreement and validity follow from the fact that the decision value of $\mathcal{B}$ is always the decision value that would be output by $\mathcal{A}'$ in the same execution. Let us show that $\mathcal{B}$ satisfies Δ-coordination.

Note first that all nodes halt no later than time $n-1$: $\mathcal{A}'$ always decides by this time and thus the valency is known to all nodes and $K_u\varphi_0$ holds for all $u \in V$. If all nodes halt exactly at time $n-1$ then clearly Δ-coordination is satisfied. Now suppose that for some $k \geq 1$, the first node u decides at time $n - k \cdot \Delta - 1$. Then $(\alpha, n - k \cdot \Delta - 1) \models K_u\varphi_k$, and thus $(\alpha, n - k \cdot \Delta - 1) \models \varphi_k$. Since $\varphi_k = E^{@(n-(k-1)\Delta-1)}\varphi_{k-1}$, we have $(\alpha, n - (k-1)\Delta - 1) \models K_v\varphi_{k-1}$ for each $v \in V$; it follows that any undecided node decides at time $n - (k-1)\Delta - 1$, no later than Δ rounds after u. □

The Δ-ladder requires nodes to keep track of information about the full dynamic graph, and to evaluate complex knowledge criteria; the clear-majority protocol uses less tight rules, but they are simpler and easier to evaluate. In general, any approximation for the knowledge criteria above can be used, as long as the same approximation is applied consistently at each decision point $n - k \cdot \Delta - 1$.

Approximate Δ-Ladder. Let $\mathcal{A}$ be an eventual consensus algorithm with round complexity at most $n-1$, let $k_{\max} \in \mathbb{N}$, and fix a collection $\{\Phi_u^{k,v}\}_{u \in V, k \in [k_{\max}], v \in \{0,1\}}$ of *local* knowledge formulas, such that u can evaluate the satisfaction of $\Phi_u^{k,v}$ based on its local state. These formulas represent the decision rules: if $\Phi_u^{k,v}$ holds at time $n - k \cdot \Delta - 1$, then node u halts at this time and outputs v. The formulas must satisfy:

(a) **Consistency:** For every u, α and t,

$$(\mathcal{R}(\mathcal{A}), \alpha, t) \models \left(\Phi_u^{0,0} \to \mathsf{val}_{\mathcal{A}}(0)\right) \quad \text{and} \quad (\mathcal{R}(\mathcal{A}), \alpha, t) \models \left(\Phi_u^{0,1} \to \mathsf{val}_{\mathcal{A}}(1)\right).$$

In other words, whenever $\Phi_u^{0,v}$ is satisfied for $v \in \{0,1\}$, the current execution must be v-valent under algorithm $\mathcal{A}$.

(b) **Timeliness:** For all executions α,

$$(\mathcal{R}(\mathcal{A}), \alpha, n-1) \models \Phi_u^{0,0} \vee \Phi_u^{0,1}.$$

That is, at time $n-1$, for some $v \in \{0,1\}$, $\Phi_u^{0,v}$ must hold, so node u knows that the current run is v-valent for $\mathcal{A}$.

(c) **Coordination:** For every $1 \leq k \leq k_{\max}$ and $v \in \{0,1\}$, if $(\mathcal{R}(\mathcal{A}), \alpha, n - k \cdot \Delta - 1) \models \Phi_u^{k,v}$, then

$$(\mathcal{R}(\mathcal{A}), \alpha, n - (k-1)\Delta - 1) \models \bigwedge_{w \in V} \Phi_w^{k-1,v}.$$

That is, whenever $\Phi_u^{k,v}$ holds for some k and v at time $n - k \cdot \Delta - 1$, we are guaranteed that at time $n - (k-1)\Delta - 1$ all formulas $\Phi_w^{k-1,v}$ for $w \in V$ will also hold.

Then a protocol for Δ-coordinated consensus is given by the following: the nodes simulate algorithm $\mathcal{A}$ with their local inputs, but do not output $\mathcal{A}$'s decisions immediately. Instead, for each $k = k_{\max}, \ldots, 1$, a node u (which has not decided already) decides v at time $n - k \cdot \Delta - 1$ if $(\mathcal{R}(\mathcal{A}), \alpha, n - k \cdot \Delta - 1) \models \Phi_u^{k,v}$. (Note that because we assumed that $\Phi_u^{k,v}$ is a *local* predicate, which depends only on u's local state, $\Phi_u^{k,v}$ holds iff $K_u \Phi_u^{k,v}$ holds.)

Lemma 4.30. *For any consensus algorithm $\mathcal{A}$, collection $\left\{\Phi_u^{k,v}\right\}_{u \in V, k \in [k_{\max}], v \in \{0,1\}}$ of local knowledge formulas satisfying Consistency, Timeliness and Coordination, the approximate Δ-ladder protocol instantiated with $\left\{\Phi_u^{k,v}\right\}$ solves Δ-coordinated consensus. Moreover, the worst-case time complexity of the protocol is $n-1$ rounds.*

Proof. We must show:

- Agreement and validity: unwinding the Coordination requirement above, we see that for any $u \in V$ we have $(\mathcal{R}(\mathcal{A}), \alpha, n - k \cdot \Delta - 1) \models \Phi_u^{k,v} \Rightarrow (\mathcal{R}(\mathcal{A}), \alpha, n - (k-1) \cdot \Delta - 1) \models \Phi_u^{k-1,v} \Rightarrow \ldots \Rightarrow (\mathcal{R}(\mathcal{A}), \alpha, n-1) \models \Phi_u^{0,v}$. By the Consistency requirement, $\Phi_u^{0,v}$ implies $\mathsf{val}_{\mathcal{A}}(v)$. Therefore $\Phi_u^{k,v}$ implies $\mathsf{val}_{\mathcal{A}}(v)$.

  If node u decides v at some time $n - k \cdot \Delta - 1$, then $\Phi_u^{k,v}$ holds, and therefore the run is v-valent for $\mathcal{A}$. Since $\mathcal{A}$ is a correct consensus algorithm, agreement and validity follow.

- Δ-coordination: suppose that node u is the first to decide, and it decides v at time $n - k \cdot \Delta - 1$. Then $(\mathcal{R}(\mathcal{A}), \alpha, n - k \cdot \Delta - 1,) \models \Phi_u^{k,v}$. By the Coordination requirement we have $(\mathcal{R}(\mathcal{A}), \alpha, n - (k-1)\Delta - 1) \models \Phi_w^{k-1,v}$ for each $w \in V$, and consequently, any node that did not decide at time $n - k \cdot \Delta - 1$ decides v at time $n - (k-1)\Delta - 1$.

Note that Timeliness has so far not been used; indeed, any set of local decision rules satisfying Consistency and Coordination yields a safe Δ-coordination algorithm. However, the resulting algorithm may not terminate; for example, if we choose $\Phi_u^{k,v} =$ **false** for all u, k and v, then Consistency and Coordination hold vacuously, but nodes will never decide. Timeliness guarantees that at time $n-1$, either $\Phi_u^{0,0}$ or $\Phi_u^{0,1}$ holds, causing all nodes that have not yet decided to now decide. □

The approximate Δ-ladder allows us to use any set of decision rules $\left\{\Phi_u^{k,v}\right\}$, not just the rules we gave in the original Δ-ladder.

Example 4.31. The clear-majority algorithm is an instance of the approximate Δ-ladder, obtained by taking the consensus algorithm $\mathcal{A}$ that forwards all inputs and decides on the majority input value, together with the following decision rules:

$$\Phi_u^{k,v} = \text{ node } u \text{ received at least } \lfloor n/2 \rfloor + 1 + \binom{k}{2}\Delta \text{ copies of input value } v.$$

The proof of Lemma 4.27 shows that these rules satisfy consistency, timeliness and coordination.

Finally, let us give another instantiation of the approximate Δ-ladder, which decides quickly in graphs where all nodes hear from everyone quickly.

Dynamic Diameter-Based Protocol. Let $f : \{0, 1\}^n \to \{0, 1\}$ be any function that satisfies $f(0^n) = 0$ and $f(1^n) = 1$. Let $\mathcal{A}$ be the following (untimed) consensus algorithm: for $n - 1$ rounds, all nodes forward all inputs they have heard so far; at time $n - 1$ all nodes know the entire input assignment $\bar{x}$, and they decide $f(\bar{x})$.

We now construct a Δ-coordinated consensus algorithm based on $\mathcal{A}$. In this protocol all nodes forward the full structure of the dynamic graph that they have seen so far, as well as all inputs they are aware of. At time $t_k := n - k \cdot \Delta - 1$, a node decides $f(\bar{x})$ if it knows that the input assignment is $\bar{x} \in \{0, 1\}^n$, and it knows that there exists some D such that the dynamic graph had a dynamic diameter of at most D until time $(k - 1)D$ (where $(k - 1)D \leq n - k \cdot \Delta - 1$). That is,

$$\begin{aligned}\Phi_u^{k,v} = K_u\Big(&\exists \bar{x} \in \{0, 1\}^n\, [\mathsf{input}(\bar{x}) \wedge f(\bar{x}) = v] \\ &\wedge \quad \exists D : \forall t \in [0, (k-2)D], \forall x, y \in V\ [(x, t) \rightsquigarrow (y, t + D)]\Big),\end{aligned}$$

where $\mathsf{input}(\bar{y})$ is a formula that holds iff the input assignment is $y \in \{0, 1\}^n$. As usual, the decision points t_k are defined for $k = 1, \ldots, k_{\max}$, where $k_{\max}$ determines the earliest point at which some node can decide.

Let us show that these decision rules satisfy the requirements.

- Consistency: nodes only decide once they know that the input is $\bar{x}$, and their decision value is then $f(\bar{x})$. This is the same value that is decided at time $n - 1$ under $\mathcal{A}$, so Consistency is satisfied.

- Timeliness: the decision rule $\Phi_u^{0,v}$ for time $n - 1$ (that is, $k = 0$) asserts that node u should decide v if it knows that $\bar{x}$ is the input and $v = f(\bar{x})$, and if it knows that there exists some D such that the dynamic diameter is at most $n - 1$ up until time $(k - 1)D$.

  The first requirement holds at time $n - 1$ because $n - 1$ rounds are sufficient for all nodes to assemble the initial input assignment (by Lemma 2.21). The second requirement holds vacuously, since for $k = 0$ we obtain $(k - 1)D < 0$.

- Coordination: suppose that the rule for deciding at time $n - k \cdot \Delta - 1$ holds at node u, i.e., u knows the input assignment, and for some D, node u knows that dynamic diameter of the graph is at most D until time $(k - 1)D \leq n - k \cdot \Delta - 1$.

  If $k = 1$, then the decision rule for time $n - (k - 1)\Delta - 1 = n - 1$ holds at time $n - 1$ for all nodes, as we showed above (Timeliness).

  If $k \geq 2$, then because the dynamic diameter is at most D until time $(k - 1)D$, for any two nodes w, w' we have $(w, (k - 2)D) \rightsquigarrow (w', (k - 1)D)$. Consequently, at time $(k-1)D \leq n-(k-1)\Delta - 1 < n-(k-2)\Delta - 1$, all nodes know that the dynamic graph had diameter at most D up to time $(k-2)D$, and all nodes know the input assignment. This is exactly the criterion for deciding at time $n-(k-1)\Delta-1$. Note that the decision rules are all *stable* properties; once true, they remain true forever. Therefore when time $n-(k-1)\Delta-1$ arrives the decision rule for $k-1$ is satisfied and all undecided nodes now decide.

The value we choose for $k_{\max}$ should satisfy $(k_{\max}-1)D < n - k_{\max}\cdot\Delta - 1$, otherwise the decision rule for time $t_{k_{\max}}$ would be unsatisfiable: a node can never know at time $t_{k_{\max}} = n - k_{\max}\cdot\Delta - 1$ that the graph had diameter at most D up to time $(k_{\max}-1)D$ if time $(k_{\max}-1)D$ occurs after time $t_{k_{\max}}$. It is sufficient to choose $k_{\max} := \lfloor n/(D+\Delta)\rfloor$. For this value of $k_{\max}$, nodes can decide as early as time

$$t_{k_{\max}} = n - k_{\max}\cdot\Delta - 1 \leq n - \frac{n\Delta}{D+\Delta} + 1 - 1 = O\left(\frac{nD}{D+\Delta}\right).$$

For example, if the communication graph is always a clique ($D=2$), then the running time is slashed by a factor of Δ. Note that the algorithm does not commit in advance to some diameter D; nodes always evaluate the stopping condition with respect to all D, and check if some bound D satisfies the requirement.

Theorem 4.32. *There is a Δ-approximate algorithm that decides at time $O(nD/(D+\Delta))$ in any dynamic graph of dynamic diameter D.*

4.5 Lower Bounds on Δ-Coordinated Consensus

We now prove two lower bounds that complement the upper bounds from the previous section. The first lower bound, Theorem 4.39, shows that in static line executions, for any input assignment, no Δ-coordinated consensus algorithm terminates in fewer than $n - O(\Delta^{0.28}n^{0.72})$ rounds. This shows that the best-case performance of the Clear Majority algorithm, $n - \Theta(\sqrt{n\Delta})$ (Lemma 4.27), is "of the right type" (although it may not be optimal). The second lower bound, Theorem 4.40, shows that for any dynamic diameter $D \leq n/2$, there is a graph of dynamic diameter D in which Δ-coordinated consensus requires $\Omega(nD/(D+\Delta))$ rounds, which shows that the Dynamic Diameter-Based Protocol from Theorem 4.32 is optimal for any given D. Our lower bounds are for undirected graphs.

4.5.1 General Setup

Our lower bounds require an adaptation of the lower bound technique we used in Section 4.3 to show that common knowledge cannot be acquired by time $n-1$. Let us begin by summarizing the lower bound from Section 4.3, as it applies to simultaneous consensus. To show that simultaneous consensus cannot decide in (α, t), we showed that there exist two points (α_0, t) and (α_1, t) such that

(a) In α_0 the input to all nodes is 0, and in α_1 the input to all nodes in 1; and

(b) $(\alpha_0, t) \sim (\alpha, t) \sim (\alpha_1, t)$.

To briefly review the argument, suppose that some node decides v in (α, t). Consider a path between (α, t) and (α_{1-v}, t) in the similarity graph; denote this path by

$$(\alpha, t) = (p_0, t) \sim_{u_1} (p_1, t) \sim_{u_2} \cdots \sim_{u_\ell} (p_\ell, t) = (\alpha_{1-v}, t).$$

We show that some node decides v in (α_{1-v}, t), violating validity, by employing the following argument at each step $i = 1, \ldots, \ell$ along the path:
1. Some node w decides v in (p_i, t); therefore,
2. From simultaneity and agreement, node u_i decides v in (p_i, t); therefore,
3. Node u_i also decides v in (p_{i+1}, t), because it cannot distinguish (p_{i+1}, t) from (p_i, t).

This argument hinges on simultaneity, and we cannot employ it as-is to prove lower bounds on Δ-coordinated consensus. However, Δ-coordination allows us to make the following weaker argument:
1. Some node w decides v in (p_i, t); therefore,
2. **From $\boldsymbol{\Delta}$-coordination and agreement, node $\boldsymbol{u_i}$ decides $\boldsymbol{v}$ in $\boldsymbol{(p_i, t + \Delta)}$;**[5] therefore,
3. If $(p_i, t + \Delta) \sim_{u_i} (p_{i+1}, t + \Delta)$, node u_i also decides v in $(p_{i+1}, t + \Delta)$, because it cannot distinguish $(p_{i+1}, t + \Delta)$ from $(p_i, t + \Delta)$.

The key difference is that unlike the simultaneous case, now we have to pay for each step we take in the similarity graph: our lower bound is weakened by Δ rounds at each step.

This reasoning, applied repeatedly, yields the following lemma.

Lemma 4.33. *Let $\alpha, \alpha_0, \alpha_1$ be executions where in α_0 and α_1 all nodes receive input 0 and 1, respectively. Assume that for some $\ell \geq 1$ and time t, the following two sequences of steps (i.e., edges) exist in the similarity graph:*

$$\begin{aligned} &(\alpha, t + \Delta) \sim_{u_1} (p_1, t + \Delta), \\ &\quad (p_1, t + 2\Delta) \sim_{u_2} (p_2, t + 2\Delta), \\ &\qquad \cdots \\ &\qquad\quad (p_{\ell-1}, t + \ell\Delta) \sim_{u_\ell} (\alpha_0, t + \ell\Delta) \end{aligned}$$

and

$$\begin{aligned} &(\alpha, t + \Delta) \sim_{u'_1} (p'_1, t + \Delta), \\ &\quad (p'_1, t + 2\Delta) \sim_{u'_2} (p'_2, t + 2\Delta), \\ &\qquad \cdots \\ &\qquad\quad (p'_{\ell-1}, t + \ell\Delta) \sim_{u'_\ell} (\alpha_1, t + \ell\Delta); \end{aligned}$$

Then in any Δ-coordinated consensus algorithm, no node can decide by time t in α.

[5] Technically, there exists some $t' \leq t + \Delta$ such that u_i decides v in (p_i, t'). The essential property is that by time $t + \Delta$ node u_i has already decided v in p_i.

Proof. As explained above, if some node decides v in (α, t), we show by induction on the path length (ℓ) that some node decides v in $(\alpha_{1-v}, t+\ell\Delta)$, violating validity. □

The condition of Lemma 4.33 involves many different times, $t+\Delta, t+2\Delta, \ldots, t+\ell\Delta$. A simpler condition that still implies a lower bound can be obtained by replacing all times with the last time, $t+\ell\Delta$ (to still obtain a lower bound for time t). We show the existence of the following two walks in the similarity graph:

$$\begin{aligned}(\alpha, t+\ell\Delta) &= (p_0, t+\ell\Delta) \sim_{u_1} (p_1, t+\ell\Delta) \sim_{u_2} \cdots \\ &\sim_{u_\ell} (p_\ell, t+\ell\Delta) = (\alpha_0, t+\ell\Delta), \text{ and} \\ (\alpha, t+\ell\Delta) &= (p'_0, t+\ell\Delta) \sim_{u'_1} (p'_1, t+\ell\Delta) \sim_{u'_2} \cdots \\ &\sim_{u'_\ell} (p'_\ell, t+\ell\Delta) = (\alpha_1, t+\ell\Delta).\end{aligned}$$

This only strengthens the condition, since $(\alpha, t) \sim_u (\alpha', t)$ implies $(\alpha, t') \sim_u (\alpha, t')$ for all $t' \leq t$. Thus the existence of these walks is sufficient to apply Lemma 4.33.

Corollary 4.34. *Let α, α_0, and α_1 be executions where in α_0 and α_1 all nodes receive input 0 and 1, respectively. Assume that for some $\ell \geq 1$ and time $t \geq \ell\Delta$, point (α, t) is at distance at most ℓ from both (α_0, t) and (α_1, t) in the similarity graph. Then in α, no node can decide by time $t - \ell\Delta$.*

When a full-information protocol is used, all information about the input becomes common knowledge at time $n-1$; therefore we cannot hope to have $t + \ell\Delta > n-1$ when we apply Lemma 4.33 or its simplified version. In order to maximize t and obtain the strongest possible lower bound, we must minimize ℓ; that is, we must find short walks in the similarity graph. Since our ultimate goal is to span between α and two runs where the inputs are 0 and 1 (respectively), the walk should allow us to flip the inputs of as many nodes as possible in each step.

4.5.2 Lower Bound For Static Paths

We now apply the strategy described above to obtain an $n - O(\Delta^{0.28} n^{0.72})$ lower bound in static paths (i.e., line graphs) of length n. A path is a natural candidate for proving strong lower bounds: we can flip the inputs of nodes at one end of the path, and the nodes at the other end do not find out for a long time.

Proposition 4.35. *If P is a static line graph over n nodes $u_1, \ldots, u_n$ (in this order), and I, I' are input assignments that differ only in the inputs to nodes $u_1, \ldots, u_m$ for $m \leq n$, then $((P, I), n-m-1) \sim_{u_n} ((P, I'), n-m-1)$.*

Proof. Node u_n, at the end of the path, does not hear from any node $u_1, \ldots, u_m$ by time $n-m-1$. Therefore $((P, I), n-m-1) \sim_{u_n} ((P, I'), n-m-1)$. □

However, to use Lemma 4.33, we must be able to flip the inputs of *all* the nodes in the network, not just the nodes at the end of the path. Thus, we start with some path $u_1, \ldots, u_n$, and flip the inputs in some prefix $u_1, \ldots, u_p$ of the path (where p is a parameter that will be fixed later). Node u_n cannot distinguish the two executions until time roughly $n-p$. Then we find a short walk in the similarity graph from our original path $u_1, \dots, u_n$ to a new path, $u_{p+1}, \dots, u_{n+p}$, i.e., we preserve the order of nodes, but we remove edge $\{u_{n+p}, u_{n+p+1}\}$ and instead put in $\{u_n, u_1\}$. (Here and in the sequel, node indices are given modulo n.) The effect is to rotate the path, so that now it starts at u_{p+1}. Now we can flip the inputs of nodes $u_{p+1}, \dots, u_{2p}$; node $u_p = u_{n+p}$, located at the end of the new path, cannot detect this change until time roughly $n-p$. We require at most $\lceil n/p \rceil$ such steps to flip any input assignment into either the all-zero or the all-one input assignment.

The strength of the lower bound is determined by the length of the walk from one path, $u_{i\cdot p+1}, \dots, u_{n+i\cdot p}$, to the next path, $u_{(i+1)\cdot p+1}, \dots, u_{n+(i+1)\cdot p}$. To construct the walk we use an intricate recursion. During the walk between paths we do not change any input values; in the sequel we focus on the dynamic graph and assume some input for all the executions we consider. As we did in Section 4.3, we simplify our notation by representing points as (G, t) where G is the dynamic graph under consideration.

Let $P_k := u_{k+1}, \dots, u_{n+k}$ denote the path starting at node u_{k+1}, and let C denote the cycle $u_1, \dots, u_n, u_1$. It is convenient to use the cycle C to bridge between paths: we cannot remove any edge of a path without violating connectivity, but a cycle is 2-vertex connected, so we can drop any of its edges. Intuitively, to move from a path P_k to a different path P_{k+s} (for $s \neq 0$), we first close P_k to form the cycle C, then drop edge $\{v_{k+s}, v_{k+s+1}\}$ to obtain P_{k+s}.

The following lemma shows how we can move from a path to the cycle while ensuring that some node cannot distinguish the two executions: it represents an intermediate step which will be used later to move between two paths. Our recursion works on increasingly long suffixes, so Lemma 4.36 assumes that some suffix $[t-b, t]$ of the execution has already been tranformed from path P_k to path P_{k+s}. We now wish to reach further into the past, to rounds $[t-b-a, t-b]$, and transform them from P_k into the cycle C, in preparation for further changes. The suffix $[t-b, t]$, where the graphs in the two executions are P_k and P_{k+s} respectively, will help cut off the spread of information about the change we make in rounds $[t-b-a, t-b]$ (see Fig. 4-1). We are able to make the change only if the lengths a, b of the segments we are working on satisfy $a+b < n-s$, otherwise indistinguishability cannot be maintained up to time t.

Lemma 4.36. *Let $k \in \{0, \dots, n-1\}$, $s \in \mathbb{Z}$, and let $0 \le a \le |s|$ and $b \ge 0$ satisfy $a+b < n-s$. Fix a time $b < t \le n-1$. Consider two graphs G, G' that agree on the first $\max\{0, t-b-a\}$ rounds, such that*

- *In rounds $r \in [t-b-a+1, t-b]$, $G(r) = C$ and $G'(r) = P_k$;*
- *In rounds $r \in [t-b+1, t]$, $G(r) = G'(r) = P_{k+s}$.*

Then $(G, t) \sim_{u_{k+s-1}} (G', t)$.

Proof. Assume that $s \ge 0$ (the other case is symmetric). At each time $r = (t-b-a+1)+i$ for $i = 1, \dots, a$, only nodes $u_{k-i}, \dots, u_{k+i+1}$ have learned of the missing edge, $\{u_k, u_{k+1}\}$, and only these nodes can distinguish G from G'. Thus, at time $t-b$, only nodes $u_{k-a}, \dots, u_{k+a+1}$ can distinguish G from G'. Next, both graphs switch to P_{k+s} where the distance between any node $u_{k-a}, \dots, u_{k+a+1}$ and node $u_{k+s-1} = u_{k-(n-s+1)}$ is at least $n-s-a$. Since G and G' are identical from this point on, only the nodes

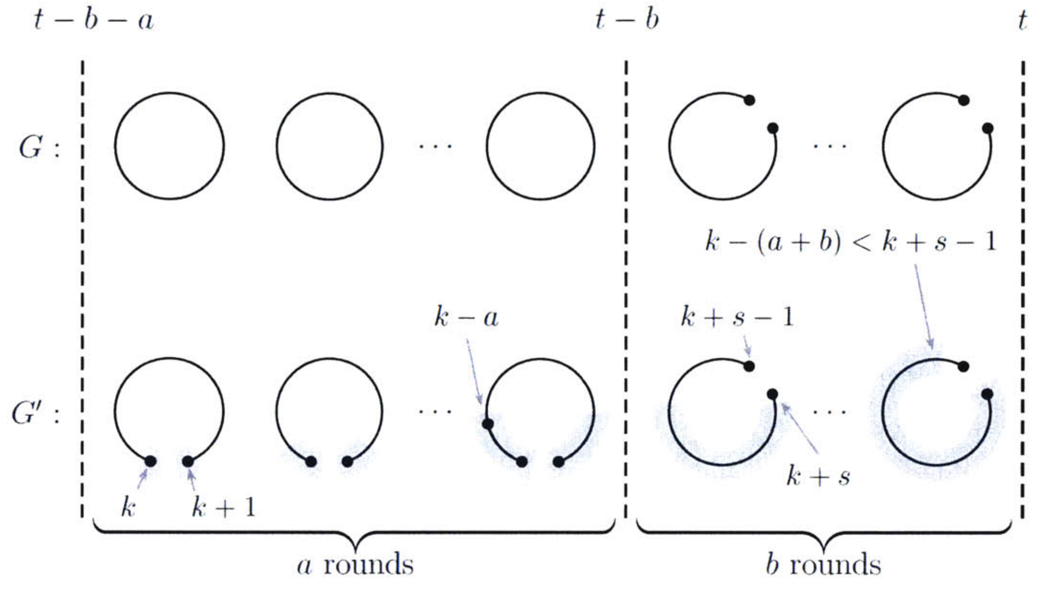


Figure 4-1: Illustration of Lemma 4.36. The shaded arcs indicate which nodes can distinguish G from G'. Switching from P_k to P_{k+s} "cuts off" the spread of information about the missing edge, and prevents it from reaching node $k + s - 1$.

that can distinguish G from G' at time $t - b$ are able to "spread information" about the difference between G and G'. Since $b < n - s - a$, by time t node u_{k+s-1} does not hear from any node that can distinguish G from G'; hence node u_{k+s-1} cannot distinguish G from G' by time t. □

Our eventual goal is to transform the entire execution from one path P_k to another path P_{k+r} Next, we show how to use Lemma 4.36 to recursively transform increasingly long suffixes of the execution from P_k to P_{k+r}. Lemma 4.36 requires us to have a suffix of the execution that is already transformed from path P_k to another path P_{k+s}, in order to cut off information propagation and maintain indistinguishability; the path P_{k+s} required by Lemma 4.36 is *not* the path P_{k+r} that we are aiming for, only an intermediate step we must go through in order to apply the lemma. Therefore the structure of the recursion is as follows (for parameters a, b, s, s' that will be fixed in the lemma below):

1. Recursively transform a suffix $[t-b, t]$ of the execution from P_k to the path P_{k+s} required for Lemma 4.36;

2. Apply Lemma 4.36 to close the path P_k into the cycle C in rounds $[t-b-a, t-b]$;

3. Recursively transform the suffix $[t - b, t]$ into a new path $P_{k+s'}$, in preparation for another application of Lemma 4.36;

4. Apply Lemma 4.36 to open the cycle C and obtain P_{k+r} in rounds $[t-b-a, t-b]$;

5. Finally, recursive transform the suffix $[t-b,t]$ from $P_{k+s'}$ into our desired path P_{k+r}.

Following the last step, the entire suffix $[t-b-a,t]$ of the execution is the path P_{k+r}. See Fig. 4-2 for an illustration (in the figure, "I.H." stands for "induction hypothesis").

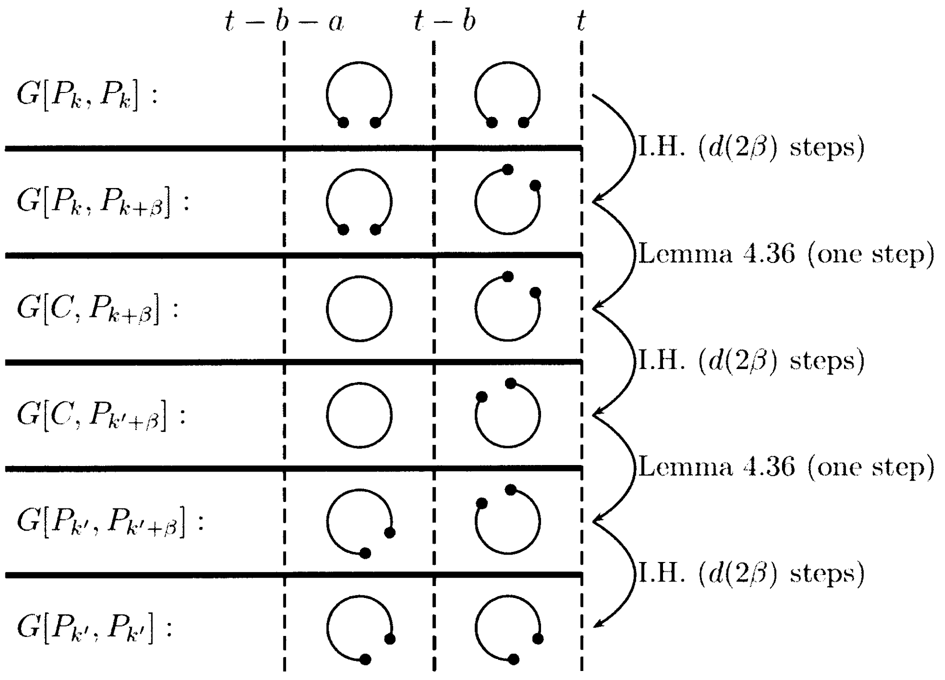


Figure 4-2: The recursion from Lemma 4.37. The two graphs shown for each step represent the communication graph for rounds $t-b-a,\dots,t-b$ and for rounds $t-b+1,\dots,t$.

Let $d((G,t),(G',t))$ denote the distance between (G,t) and (G',t) in the similarity graph.

Lemma 4.37. *Fix a time* $0 \le t \le n-1$ *and a value* $1 \le p \le n-1$. *Let* G, G' *be dynamic graphs that agree up to time* $t-(n-1-p)$, *such that in rounds* $r \in [t-(n-1-p)+1,t]$, $G(r) = P_k$ *and* $G'(r) = P_{k'}$ *(for some* k,k'*). Then* $d((G,t),(G',t)) \le 9(n/p)^{\log_2 3}$.

Proof. Define $\ell_p := \lceil \log_2(n/p) \rceil$. We show by induction on ℓ_p that $d((G,t),(G',t)) \le 3^{\ell_p+1}-1$. The claim then follows, because

$$3^{\ell_p+1}-1 \le 3^{\log_2(n/p)+2} = 9\left(\frac{n}{p}\right)^{\log_2 3}.$$

Let us denote $d_p := 3^{\ell_p}+1$. Note that we are transforming the suffix $[t-(n-1-p)+1,t]$ of the execution; hence, smaller values of p (or equivalently, larger values of ℓ_p) are "harder" because they require us to transform a longer suffix.

The induction base is immediate: if $\ell_p = \lceil \log_2(n/p) \rceil = 0$, then $p \geq n$. Since we know that G and G' agree up to time $t - (n-1-p) \geq t$, we have $(G,t) = (G',t)$, and therefore $d((G,t),(G',t)) = 0$.

For the induction step we use Lemma 4.36. Set $a = p$ and $b = n - 1 - 2p$. For convenience, let us introduce the following notation: given static graphs H_1, H_2, let $G[H_1, H_2]$ be the dynamic graph defined by

$$G[H_1,H_2](r) := \begin{cases} G(r) & r \leq t-(n-1-p), \\ H_1 & t-(n-1-p) < r \leq t-b, \\ H_2 & t-b < r \leq t. \end{cases}$$

Since $b = n-1-2p$ and $\ell_{2p} = \ell_p - 1$, the induction hypothesis shows that for any graph H and for any two paths $P_q, P_{q'}$ we have $d((G[H,P_q],t),(G[H,P_{q'}],t)) \leq d(2p)$. Further, Lemma 4.36 shows that $d((G[P_q,P_{q+p}],t),(G[C,P_{q+p}],t)) = 1$ for any q (because these points are indistinguishable to some node). Thus, we construct the following walk:

$$\begin{aligned}(G,t) = (G[P_k,P_k],t) &\overset{d(2p)}{\underset{\text{I.H.}}{\rightarrow}} (G[P_k,P_{k+p}],t) \underset{\text{Lem. 4.36}}{\overset{1}{\rightarrow}} \\ (G[C,P_{k+p}],t) &\overset{d(2p)}{\underset{\text{I.H.}}{\rightarrow}} (G[C,P_{k'+p}],t) \underset{\text{Lem. 4.36}}{\overset{1}{\rightarrow}} \\ (G[P_{k'},P_{k'+p}],t) &\overset{d(2p)}{\underset{\text{I.H.}}{\rightarrow}} (G[P_{k'},P_{k'}],t) = (G',t).\end{aligned}$$

The length of the walk is at most $3d(2p) + 2 = 3(3^{\ell_p} - 1) + 2 = 3^{\ell_p+1} - 1$. $\square$

We showed in Lemma 4.37 that we can move from a path P_k to the next path, P_{k+p}, in few steps (specifically, in $9(n/p)^{\log_2 3}$ steps). To complete the lower bound, we must use the lemma to completely transform the input assignment into the all-zero or all-one input assignment:

Lemma 4.38. *Let $\alpha = (P_1, I)$ be an execution, and let I_0, I_1 be the all-zero and all-one input assignments, respectively. Let $p \geq 0$, and define $t := n - 2p - 1$. Then for both $v = 0$ and $v = 1$,*

$$((P^1, I), t+p) \sim^{\ell} ((P^{n-p}, I_v), t+p),$$

where

$$\ell = O\left(\left(\frac{n}{p}\right)^{1+\log_2 3}\right).$$

Proof. For $v \in \{0,1\}$, we span between $((P^1,I),t+p)$ and $((P^{n-p},I_v),t+p)$ by repeating the following steps $O(n/p)$ times:

1. Flip the inputs of the leftmost p nodes on the path to v in one move. By Proposition 4.35, the rightmost node on the path cannot distinguish the previous execution from the current one until time $t+p$, because $t = n - 2p - 1$ and the distance from

the nodes whose inputs we changed to the rightmost node on the path is $n - p$. Let J be the resulting input assignment.
2. Applying Lemma 4.37, move from our current point $((P^k, J), t+p)$ to $((P^{k+p}, J), t+p)$ in $O((n/p)^{\log_2 3})$ steps, preserving similarity at time $t + p$.

The resulting similarity chain is $((P^1, I), t+p) \sim ((P^1, J_1), t+p) \sim ((P^{p+1}, J_1), t+p) \sim ((P^{p+1}, J_2), t+p) \sim \ldots ((P^{n-p}, I_v), t+p)$, where in J_i, nodes $1 \ldots, i \cdot p$ have input v, and the remaining nodes have the same input they had in the original input assignment I. The total length of the walk is $\ell = O((n/p) \cdot (n/p)^{\log_2 3}) = O((n/p)^{1+\log_2 3})$. □

Theorem 4.39. *In the static line graph P_1 no Δ-coordinated consensus algorithm can decide by time $n - O(\Delta^{\frac{1}{2+\log_2 3}} n^{1-\frac{1}{2+\log_2 3}}) \approx n - O(\Delta^{0.28} n^{0.72})$ for any input assignment.*

Proof. Let $\alpha = (P_1, I)$ be an execution, and let I_0, I_1 be the all-zero and all-one input assignments, respectively. Let p be a parameter, and set $t := n - 2p - 1$. By Lemma 4.38 we have $((P_1, I), n - p - 1) \sim^{\ell} ((P^{n-p}, I_v), n - p - 1)$ for both $v = 0$ and $v = 1$. Thus, by Corollary 4.34, no node decides by time $t - \ell\Delta = n - p - 1 - O\left((n/p)^{1+\log_2 3}\right)$.

To obtain as strong a bound as possible, we seek to minimize the term

$$p + 1 + c \cdot \Delta \cdot \left(\frac{n}{p}\right)^{1+\log_2 3},$$

where here c is the constant in the big-O expression for the length of the walk. Taking the derivative with respect to p yields

$$1 - c\Delta(1 + \log_2 3)\left(\frac{n}{p}\right)^{\log_2 3} \cdot \frac{n}{p^2}.$$

Setting this expression equal to zero and solving for p, we obtain

$$p = c(1 + \log_2 3)\Delta^{\frac{1}{2+\log_2 3}} n^{1-\frac{1}{2+\log_2 3}} = O\left(\Delta^{\frac{1}{2+\log_2 3}} n^{1-\frac{1}{2+\log_2 3}}\right).$$

Plugging this value of p into the bound of $n - p - 1 - \ell\Delta$, we see that no node decides by time $n - O(\Delta^{\frac{1}{2+\log_2 3}} n^{1-\frac{1}{2+\log_2 3}})$. □

Recall that in Section 4.4 we gave an algorithm (the Clear Majority Algorithm) whose earliest possible decision time is $n - \Theta(\sqrt{n\Delta})$. Theorem 4.39 matches this algorithm in a qualitative sense (though it is far from tight): it shows that for any algorithm there exists an assignment under which no node can decide until time $n - O(\Delta^c n^{1-c})$ for some $c \in (0, 1)$, in our case $c = 1/(2\log_2 3) \approx 0.28$.

4.5.3 Lower Bound for Graphs of Specific Dynamic Diameter

Another algorithm we gave in Section 4.4 showed that when the dynamic diameter of the graph is D, we can decide at time $O(nD/(D + \Delta))$ (Theorem 4.32). We now show that this algorithm is asymptotically optimal for *any* input assignment.

Theorem 4.40. *For every D satisfying $4 \leq D \leq n/2$ and $\Delta \geq 1$, there is a static graph of diameter D such that for any input assignment, no node decides before time $\Omega(nD/(D+\Delta))$ in any Δ-coordinated consensus protocol.*

Proof. We construct a static graph H as follows. Let

$$k := D - 3,$$
$$k_1 := \left\lfloor \frac{k}{2} \right\rfloor, \qquad k_2 := \left\lceil \frac{k}{2} \right\rceil,$$
$$n_1 := \left\lfloor \frac{n-k}{2} \right\rfloor, \qquad n_2 := \left\lceil \frac{n-k}{2} \right\rceil.$$

Partition the nodes V into four sets V_1, V_2, X_1, X_2 of size n_1, n_2, k_1 and k_2, respectively, and fix an arbitrary ordering $V_1 = \{u_1, \ldots, u_{n_1}\}$, $V_2 = \{v_1, \ldots, v_{n_2}\}$. The edges of H are the following:

- The nodes of V_1 are connected in a clique, and the nodes of V_2 are connected in a clique;
- The nodes of $X := X_1 \cup X_2$ are connected in a path P_X, starting with the nodes of X_1 (in arbitrary order) and then the nodes of X_2 (in arbitrary order).
- The first node in P_X is connected to u_{n_1}, and the last node in P_x is connected to v_1.

Since X contains $k = D - 3$ nodes, the length of the path P_X is $D - 4$. Every node of V_1, V_2 is at distance at most 2 from an endpoint of P_x. Therefore the diameter of H is D. Our goal is to show that in H, for every input assignment, no node can decide until time $\Omega(nD/(D+\Delta))$. To do this, we gradually transform H into a static path, where we can flip many inputs at once without some node distinguishing the difference. Then we apply Corollary 4.34 to show that no node can decide quickly.

Formally, define

$$t_0 := \left\lfloor \frac{n(D-3)\Delta}{2(D+\Delta)} \right\rfloor.$$

Our goal now is to incrementally transform the first $\approx t_0$ rounds of the static graph H into the static path P using a short similarity chain. We work in increments of k_2 rounds, transforming first the first k_2 rounds of H into P, then the next k_2 rounds, and so on. For each $j = 1, \ldots, \lfloor t_0/k_2 \rfloor$, let $t_j := j \cdot k_2$, and define a graph G_j as follows:

- In rounds $1, \ldots, t_j$, the communication graph is H.
- Starting from round $t_j + 1$ onwards, the communication graph is P.

We claim that for each j we have $(G_j, \lfloor n/2 \rfloor - 2) \sim^2 (G_{j+1}, \lfloor n/2 \rfloor - 2)$. To see this, define an intermediate graph G'_j, where

- In rounds $1, \ldots, t_j$, the communication graph is H (as in both G_j and G_{j+1}).

- In rounds $t_{j+1}+1, \ldots$ the communication graph is P (again, as in G_j and G_{j+1}).
- In rounds $t_j+1, \ldots, t_{j+1}$, the communication graph is H', which is a hybrid of H and P: the nodes of V_1 have their edges from P, that is, they are connected in a path ending at u_{n_1}; the nodes of V_2 have their edges from H, i.e., they are connected in a clique. The nodes of X have their edges from P and from H: in both graphs they are connected in a path P_X, connected to u_{n_1} and v_1 at its respective endpoints.

We show the following.

Claim 4.41. $(G_j, \lfloor n/2 \rfloor - 2) \sim_{u_2} (G'_j, \lfloor n/2 \rfloor - 2)$.

Proof. Up to time t_j the two graphs are identical. In rounds $t_j+1, \ldots, t_{j+1}$, the communication graph in G_j is H, while the communication graph in G'_j is H'; however, H and H' differ only on the nodes in V_1. By time $t_{j+1} = t_j + k_2$, only the nodes of $V_1 \cup X_1 \cup \{w\}$, where w is the first node of X_2, can possibly distinguish G_j from G'_j, as no other nodes are in the k_2-neighborhood of V_1 (recall that $|X_1| = k_1 = \lfloor k/2 \rfloor$ while $|X_2| = k_2 = \lceil k/2 \rceil$, and therefore $k_2 \leq |X_1|+1$). Finally, after time t_{j+1}, graphs G_j and G'_j are again identical, and in both graphs the distance from any node of $V_1 \cup X_1 \cup \{w\}$ to node u_2 is at least $k_2 + n_2 - 2 = \lceil k/2 \rceil + \lceil (n-k)/2 \rceil - 2 \geq n/2 - 2$. This distance is greater than the number of rounds between time $t_{j+1} \geq 1$ and time $\lfloor n/2 \rfloor - 2$, and therefore node u_2 cannot distinguish G_j from G'_j at time $\lfloor n/2 \rfloor - 2$. □

Claim 4.42. $(G'_j, \lfloor n/2 \rfloor - 2) \sim_{u_1} (G_{j+1}, \lfloor n/2 \rfloor - 2)$.

Proof. Similar to the previous claim. Up to time t_j the graphs are identical, and in rounds $t_j+1, \ldots, t_{j+1}$ they differ only on the nodes of V_2. At time t_{j+1} only the nodes of $V_2 \cup X_2$ can distinguish G'_j from G_{j+1}, because these nodes comprise the k_2-neighborhood of V_2. Finally, after time t_{j+1} the graphs are identical, and the distance of any node in $V_2 \cup X_2$ from node u_1 is at least $n_1 + k_1 \geq (n-k)/2 - 1 + k/2 - 1 = n/2 - 2$. Therefore node u_1 cannot distinguish G'_j from G_{j+1} until time $\lfloor n/2 \rfloor - 2$. □

Together the two claims show that $(G_j, \lfloor n/2 \rfloor - 2) \sim^2 (G_{j+1}, \lfloor n/2 \rfloor - 2)$, and putting the entire chain together we obtain

$$(G_0, \lfloor n/2 \rfloor - 2) \sim^{2\lfloor t_0/k_2 \rfloor} (G_{j_{\lfloor t_0/k_2 \rfloor}}, \lfloor n/2 \rfloor - 2). \tag{4.5.1}$$

For convenience let us denote $G_* := G_{j_{\lfloor t_0/k_2 \rfloor}}$.

Note that G_0 is just the static path P. If v_0 is the all-zero input assignment, for any input assignment I we have $((G_0, I), \lfloor n/2 \rfloor - 2) \sim^2 ((G_0, v_0), \lfloor n/2 \rfloor - 2)$: to get from (G_0, I) to (G_0, v_0) we first flip the inputs of the first $\lfloor n/2 \rfloor$ nodes on the path from their values in I to 0 in one step, and then flip the values of the other $\lceil n/2 \rceil$ nodes to 0 in one more step. In both steps, one endpoint of the path cannot distinguish the two executions until time $\lfloor n/2 \rfloor - 2$, because its distance from the nodes whose inputs we changed is at least $\lfloor n/2 \rfloor$. Similarly we can show that

$((G_0, I), \lfloor n/2 \rfloor - 2) \sim^2 ((G_0, v_1), \lfloor n/2 \rfloor - 2)$, where v_1 is the all-one input assignment. Combining these four steps with the chain from (4.5.1) above yields the following:

$$\begin{aligned}((G_0, v_0), \lfloor n/2 \rfloor - 2) \sim^2 ((G_0, I), \lfloor n/2 \rfloor - 2) &\sim^{2\lfloor t_0/k_2 \rfloor} ((G_*, I), \lfloor n/2 \rfloor - 2) \\ &\sim^{2\lfloor t_0/k_2 \rfloor} ((G_0, I), \lfloor n/2 \rfloor - 2) \sim^2 ((G_0, v_1), \lfloor n/2 \rfloor - 2).\end{aligned}$$

We now appeal to Corollary 4.34: using values of $t = \lfloor n/2 \rfloor - 2$ and $\ell = 4\lfloor t_0/k_2 \rfloor + 4$, the corollary shows that in G_*, if c is sufficiently small, no node decides until time

$$\begin{aligned} t - \ell\Delta &= \left\lfloor \frac{n}{2} \right\rfloor - 2 - 2\left(\left\lfloor \frac{\left\lfloor \frac{n(D-3)\Delta}{2(D+\Delta)} \right\rfloor}{\left\lceil \frac{D-3}{2} \right\rceil} \right\rfloor + 4 \right) \\ &\geq \frac{n}{2} - 3 - 2\left(\frac{\frac{n(D-3)\Delta}{2(D+\Delta)}}{\frac{D-3}{2}} + 4 \right) = \frac{n}{2} - \left(\frac{n\Delta}{2(D+\Delta)} \right) - 11 \\ &= \frac{nD + n\Delta - n\Delta}{2(D+\Delta)} - 11 = \frac{nD}{2(D+\Delta)} - 11 = \Omega\left(\frac{nD}{D+\Delta} \right). \end{aligned}$$

We are almost done, but one important detail remains: the graph G_* for which we have a lower bound is not a static graph, and it does not have a small dynamic diameter. We must show that the lower bound also holds in the static graph H, which does have a dynamic diameter of D. However, we are not far from our goal, since G_* is identical to H for the first $k_2\lfloor t_0/k_2 \rfloor \geq t_0 - k_2$ rounds by construction. We have shown that in G_*, no node decides until time $nD/(2(D + \Delta)) - 11$. For sufficiently large n, this time is no smaller than $nD/(3(D + \Delta))$, and hence for any constant $c \geq 3$ no node decides until time $t_c := nD/(c(D + \Delta)) = \Omega(nD/(D + \Delta))$. If we can find a value of c for which $t_c \leq t_0 - k_2$, then we have shown the lower bound for the static graph H, because G_* and H are identical up to time $t_0 - k_2$ (and in particular no node can distinguish them). For sufficiently large n, a value of $c \geq 18$ suffices, because it satisfies

$$t_c = \frac{nD}{c(D+\Delta)} \leq t_0 - k_2 = \left\lfloor \frac{n(D-3)\Delta}{2(D+\Delta)} \right\rfloor - \left\lceil \frac{D-3}{2} \right\rceil.$$

□

Part II

The Communication Complexity of Distributed Computation in Directed Networks

Chapter 5

Directed Networks and Two-Player Communication Games

In this part of the thesis we turn our attention to networks that are not dynamic, but still retain an important property that distinguishes wireless networks from wired networks: they are potentially *asymmetric*. In a wired network, nodes can communicate with each other directly only if they are linked by a cable; the communication medium is inherently symmetric, and wired networks typically exploit this property and allow bidirectional communication between every pair of adjacent nodes. In contrast, wireless communication is not symmetric by nature: diffraction, absorption and multi-path propagation effects can lead to situations where two terminals transmit at the same power, but only one of them receives from the other with a signal-to-noise ratio high enough to decode the transmission. In addition, since power consumption is of greater concern in wireless networks, nodes may wish to transmit at different power levels in order to conserve energy, and this is another source of asymmetry. In the next two chapters we will study the effects of asymmetry on basic distributed computation tasks in static networks. In Chapter 6 we study traditional tasks such as estimating the size of the network and computing sensitive functions. In Chapter 7 we introduce a new problem, *task allocation*, study its communication complexity, and use it to obtain a lower bound on finding a rooted spanning tree in small-diameter networks.

5.1 Directed Broadcast Networks

The model we use to study static radio networks is a special case of the dynamic graph model introduced in Chapter 2: we simply assume now that the network graph remains static throughout the execution. The model retains its other features, including full synchrony, directed communication links and communication by local broadcast.

Formally, a directed network is modelled as a strongly-connected directed graph $G = (V, E)$, where V is a set of nodes drawn from some large UID space $\mathcal{U}$, and $E \subseteq V^2$ is a set of directed communication links. In Chapter 2 the dynamic graph was generated on-the-fly by a worst-case adversary; since we now consider static networks, the adversary's role is reduced to choosing the static graph G, and we assume that it does so before the execution begins (i.e., the adversary is oblivious). As in Chapter 3, we are concerned with algorithms that use sub-linear sized messages: we let B denote the maximum number of bits that can be sent by a single node in one round. We call B the *messages size* or *bandwidth* of the algorithm, and we typically assume that B is sublinear, that is, $B = o(n)$.

As in previous chapters, one of the central themes in this part of the thesis will be the effect of prior knowledge on the hardness (that is, the time complexity) of solving various tasks. In Chapter 2 we introduced the *hearing from everyone task*, HF_n, to model the problem of estimating the dynamic diameter of the network (i.e., learning when a particular node has been causally influenced by all other nodes). In a static network, HF_n reduces to the following task:

Definition 5.1 (HF_n in Static Networks). *We say that nodes solve the* HF_n task *in a static graph $G = (V, E)$ if every node u halts at some time t such that $\Upsilon^t(u) = V$.*

(Recall from Section 1.6 that $\Upsilon^d(u) = \{v \in V \mid \text{dist}(v, u) \leq d\}$.) Equivalently, the HF_n problem can be stated as: each node u should halt at a time t no smaller than u's *inwards-eccentricity*, that is, the largest distance from any node to u. This is simply the restriction of Definition 2.24 to static graphs. (Definition 2.24 defines the more general task HF_m for any m; here we are interested in the specific case where $m = n$).

If it is known in advance that the diameter of the graph is D, then a trivial solution to HF_n is simply to wait until time D and then halt. Thus, HF_n is only "interesting" when the diameter of the graph is not known in advance, and this is the context in which we will study the problem in Section 6.2.

In addition to HF_n, we will also study other tasks introduced in Part I of the thesis, such as approximate counting and computing the minimum input. The general definition of a single-shot task (see Section 2.2) remains the same for this part of the thesis. Our definitions and results concerning causality and information flow (Section 2.3) also remain in effect, as static directed graphs are a special case of the dynamic graph model from Chapter 2. We remain concerned with time complexity as the main measure of an algorithm's efficiency, and treat the message size B as an external parameter.

5.2 Two-Player Communication Complexity

In this section we review the model of two-player communication games, as well as several celebrated communication complexity lower bounds. These lower bounds will form the basis for our results in the next chapter.

5.2.1 Two-Player Communication Games

A two-player communication game over a universe $\{0, 1\}^n$ involves two players, Alice and Bob, who receive private inputs $x, y \in \{0, 1\}^n$ (respectively) and wish to compute some Boolean function $f(x, y)$ of their joint inputs. To accomplish this goal, Alice and Bob communicate some number of rounds, sending as many bits as they wish in each round, until eventually both players output $f(x, y)$. We are interested in the total number of bits exchanged between Alice and Bob in the worst case. The *deterministic communication complexity* of an algorithm is the worst-case number of bits exchanged on any input; the *deterministic communication complexity of f* is the minimum deterministic communication complexity of any deterministic protocol for computing f.

As for randomized protocols, the literature considers two types: *public-coin protocols*, where Alice and Bob have access to a shared source of randomness, and *private-coin protocols*, where each player has its own private randomness. We are interested here in randomized protocols that err with some small probability, which is fixed and independent of n; unless otherwise specified, we will assume that the error probability is $1/3$.[1] The *randomized communication complexity* of a randomized algorithm is defined as the expected number of bits exchanged in the worst case over inputs, and the *public-coin (resp. private-coin) randomized communication complexity of f* is obtained by taking the minimum over all public-coin (resp. private-coin) protocols for computing f.

The lower bounds we review below were proven for private-coin protocols. However, a result of Newman [118] shows that they can be extended to public-coin protocols without much loss:

Theorem 5.2 ([118]). *Any public-coin protocol can be transformed into a private-coin protocol that uses at most $O(\log n)$ additional communication bits.*

Since the lower bounds we are interested in are super-logarithmic in n, this transformation will allow us to apply them to public-coin protocols as well as to private-coin protocols.

5.2.2 Lower Bounds in Communication Complexity

The two-player communication complexity model was introduced by Yao in [143], and has since found application in many areas, including circuit lower bounds, streaming algorithms and data structures. One of the most widely applied results is the $\Omega(n)$ lower bound on the randomized communication complexity of SET-DISJOINTNESS, where the players must determine whether their input sets intersect or not:

Definition 5.3 (Set Disjointness, DISJ_n). *In the* Set Disjointness *problem, denoted* DISJ_n, *Alice and Bob receive sets $X, Y \subseteq [n]$ (represented as binary n-bit vectors), and must determine whether $X \cap Y = \emptyset$. Formally,*

$$f_{\text{DISJ}_n}(X, Y) = \begin{cases} 0 & \text{if } X \cap Y \neq \emptyset, \\ 1 & \text{otherwise.} \end{cases}$$

[1]As usual, the exact constant does not matter; we can achieve an arbitrarily small constant probability of error by repeating the protocol a constant number of times.

In the sequel we use upper-case letters to denote sets, and small-case letters to denote their respective characteristic vector (e.g., we use x, y to denote the characteristic vectors of Alice and Bob's inputs X, Y).

A trivial solution to Set Disjointness is to have Alice send her entire input over to Bob, who then compares it with his input and sends back one bit indicating whether to accept or reject. In a celebrated result, this naive strategy has been proven to be asymptotically optimal, even for randomized protocols.

Theorem 5.4 ([80, 128]). *The randomized communication complexity of* DISJ_n *is* $\Omega(n)$.

This bound was first proven by Kalyanasundaram and Schnitger in [80], and the proof was simplified by Razborov in [128]. The communication complexity of SET-DISJOINTNESS with sets of restricted size was studied, for example, in [70, 102]. For both deterministic and randomized protocols, the complexity of SET-DISJOINTNESS with sets of size k for $k \leq cn$ (where c is a sufficiently small constant) is known to be $\Omega(k)$.

We are also interested here in a relaxed variant of Set Disjointness, where the players are only required to distinguish sets that do not intersect from sets that have a large intersection.

Definition 5.5 (Gap Set Disjointness, $\text{GAP-DISJ}_{n,g}$). *In the* Gap Set Disjointness *problem, denoted* $\text{GAP-DISJ}_{n,g}$*, the players are given sets* $X, Y \subseteq [n]$*, with the promise that either* $X \cap Y = \emptyset$ *or* $|X \cap Y| \geq g$*. The players must determine which of these cases holds.*

When the gap g is large with respect to n (e.g., $g = cn$ for some constant $c \in (0, 1)$), $\text{GAP-DISJ}_{n,g}$ is quite easy for randomized algorithms: one can distinguish sets with a large intersection from disjoint sets by sampling elements at random and testing whether they belong to the intersection. However, for deterministic protocols the problem remains hard even with a linear gap. This fact appears to be folklore in the communication complexity community; for the sake of completeness, and to establish the exact constant in the gap, we include a proof.

Theorem 5.6 (folklore). *For any constant* $\varepsilon \in (0, 1/2)$*, the deterministic communication complexity of* $\text{GAP-DISJ}_{n,(1/2-\varepsilon)n}$ *is* $\Omega(n)$.

The proof uses a classical result in extremal set theory by Frankl and Rödl [55]:

Theorem 5.7 ([55]). *For any constant* $\rho \in (0, 1/2)$ *there is a constant* $\delta(\rho) > 0$ *such that for any even distance* $d \in [\rho n, (1-\rho)n]$*, if* $C \subseteq \{0,1\}^n$ *is a set such that for all* $x, y \in C$ *we have* $\Delta(x, y) \neq d$*, then* $|C| \leq 2^{n(1-\delta(\rho))}$.

Proof of Theorem 5.6. The lower bound can be shown by the *tiling method* [101, 7]. In this approach we view a protocol for $\text{GAP-DISJ}_{n,g}$ as operating on a matrix $M \in \{0, 1\}^{\{0,1\}^n \times \{0,1\}^n}$, where each element $M(x, y)$ corresponds to the desired output value

$f(x,y)$:

$$f(x,y) = \begin{cases} 1 & \text{if } |X \cap Y| \geq g, \\ 0 & \text{if } X \cap Y = \emptyset, \text{ and} \\ * & \text{otherwise.} \end{cases}$$

(Here $*$ stands for "don't care", i.e., pairs on which any output is permissible.)

A *combinatorial rectangle* is a subset $R = A \times B$ of matrix entries, where $A, B \subseteq \{0,1\}^n$ are sets of row and column indices, respectively. A rectangle is *monochromatic* if it does not contain both 0 and 1 entries. Before the protocol begins, each player knows only his or her own input, so Alice considers all instances $\{x\} \times \{0,1\}^n$ as possible and Bob considers $\{0,1\}^n \times \{y\}$ as possible. An execution of the protocol induces a partition of the matrix M into combinatorial rectangles: for example, if the first bit is sent by Alice, this partitions the matrix into two rectangles $A_0 \times \{0,1\}^n, A_1 \times \{0,1\}^n$, where each $A_i \subseteq \{0,1\}^n$ is the set of inputs x that would cause Alice to send bit i. If Alice also sends the second bit, this refines the partition into four rectangles $A_{00} \times \{0,1\}^n, A_{01} \times \{0,1\}^n, A_{10} \times \{0,1\}^n, A_{11} \times \{0,1\}^n$, and so on. Similarly, each bit sent by Bob refines the partition and doubles the number of rectangles. At the end of the protocol, if a total of k bits are sent by both players, the matrix M is partitioned into 2^k combinatorial rectangles, where each rectangle $R = A \times B$ contains only instances (x,y) that lead to the same trace of the protocol, that is, the same k bits being sent during the protocol by the players.

Each rectangle $R = A \times B$ must be *monochromatic*: for all $x_1, x_2 \in A$ and $y_1, y_2 \in B$, if $f(x_1,y_1) \neq *$ and $f(x_2,y_2) \neq *$, then we must have $f(x_1,y_1) = f(x_2,y_2)$. This is because Alice's output is determined only by her input $x \in A$ and the communication pattern she observes during the protocol, which is the same for all $y_1, y_2 \in B$; and similarly, Bob's output given input $y \in B$ is the same for all $x_1, x_2 \in A$. It follows that $f(x_1,y_1) = f(x_1,y_2) = f(x_2,y_2)$. Thus, the protocol induces a *monochromatic tiling* of M: a partition of $\{0,1\}^n \times \{0,1\}^n$ into monochromatic combinatorial rectangles. A lower bound on the communication complexity can be obtained by bounding the size of a monochromatic tiling of M; if we can show that r rectangles are necessary to cover M, then the communication complexity is at least $\log r$.

Now suppose we are given a monochromatic tiling $\mathcal{T}$ for $\textsc{Gap-Disj}_{n,(1/2-\varepsilon)n}$, and let us focus on instances of the form $(X, \overline{X})$. Given a rectangle $R \subseteq \{0,1\}^n \times \{0,1\}^n$, let $\text{support}(R) := \{x \in \{0,1\}^n \mid (X, \overline{X}) \in R\}$. Since $\mathcal{T}$ covers M, we must have $\bigcup_{R \in \mathcal{T}} \text{support}(R) = \{0,1\}^n$, or in other words $\sum_{R \in \mathcal{T}} |\text{support}(R)| = 2^n$. We will show the following:

Claim. *For some constant $\delta < 1$, for each $R \in \mathcal{T}$ it holds that* $|\text{support}(R)| \leq 2^{\delta n}$.

This will allow us to conclude that $|\mathcal{T}| \geq 2^{(1-\delta)n}$, and consequently the communication complexity of $\textsc{Gap-Disj}_{n,(1/2-\varepsilon)n}$ is at least $(1-\delta)n$.

To prove the claim, suppose that $x, y \in \text{support}(R)$ for some rectangle $R \in \mathcal{T}$. Clearly, since $X \cap \overline{X} = Y \cap \overline{Y} = \emptyset$, the "color" of the rectangle is 0: $f(x,\overline{x}) = f(y,\overline{y}) = 0$. Since R is monochromatic, we must also have $f(x,\overline{y}) = f(\overline{x},y) \in \{0,*\}$. This implies that $\Delta(x,y) < (1-2\varepsilon)n$ (where $\Delta(z,w)$ denotes the Hamming distance between z and w, that is, the number of bits in which the two vectors differ): otherwise either $|X\backslash Y| \geq (1/2-\varepsilon)n$ or $|Y\backslash X| \geq (1/2-\varepsilon)n$, and hence either $|X\cap\overline{Y}| \geq (1/2-\varepsilon)n$ or $|\overline{X} \cap Y| \geq (1/2-\varepsilon)n$. In both cases the rectangle is not monochromatic.

To bound the size of the rectangle we use Theorem 5.7. We have shown that any two sets $x, y \in R$ must have $\Delta(x, y) \leq (1-2\varepsilon)n$. In particular, if we choose $d \geq (1-2\varepsilon)n$, no two sets $x, y \in R$ have $\Delta(x, y) = d$. Fix $\rho := \min\{\varepsilon, 1/4\}$. Since $\rho < 2\varepsilon$ we have $1 - 2\varepsilon < 1 - \rho$, and hence for sufficiently large n there exists an even value $d \in (\max\{\rho, (1-2\varepsilon)\}\, n, (1-\rho)n]$. Applying the theorem, we see that the size of support(R) is bounded by $|\operatorname{support}(R)| \leq 2^{n(1-\delta(\rho))}$ where $\delta(\rho)$ is a positive constant. This completes the proof. $\square$

More recently, the Gap-Hamming-Distance (GHD) problem has also drawn attention in the complexity community. In GHD the players each receive a binary vector and must determine whether the Hamming distance of their vectors exceeds a threshold $n/2 + g$, or whether it is below the threshold $n/2 - g$, where the parameter g is called the *gap*.

Definition 5.8 (Gap Hamming Distance, $\mathrm{GHD}_{n,g}$). *In the* Gap Hamming Distance *problem, denoted* $\mathrm{GHD}_{n,g}$, *the players receive vectors* $x, y \in \{0,1\}^n$ *and must determine whether the Hamming distance* $\Delta(x, y)$ *satisfies* $\Delta(x, y) > n/2 + g$ *or whether* $\Delta(x, y) \leq n/2 - g$. *Formally,*

$$f_{\mathrm{GHD}_{n,g}} = \begin{cases} 0 & \text{if } \Delta(x,y) > n/2 + g, \\ 1 & \text{if } \Delta(x,y) \leq n/2 - g, \text{ and} \\ * & \text{otherwise,} \end{cases}$$

where $*$ *indicates that any answer is permissible.*

The GHD problem was first studied in [74], where it was shown that lower bounds on the communication complexity of GHD with a gap of $\Theta(\sqrt{n})$ yield lower bounds on the space complexity of counting the number of distinct elements in a data stream.

The one-way communication complexity of GHD was shown to be $\Omega(n)$ in [142]; subsequent work (e.g., [26]) obtained lower bounds for the two-way communication complexity with a constant number of rounds. Finally, in [29], Chakrabarti and Regev characterized the unrestricted two-way communication complexity of GHD completely. The lower bound was simplified by Vidick [140], and further simplified by Sherstov [135].

Characterizing the randomized communication complexity of GHD remained an open problem for a long time after its introduction in [74] (for the case $g = \sqrt{n}$, which is in some sense the most interesting setting[2]) until in [29], Chakrabarti and Regev proved the following lower bound. Their proof was later simplified in [140, 135].

Theorem 5.9 ([29]). *For any* $g \leq n$, *the private-coin randomized communication complexity of* $\mathrm{GHD}_{n,g}$ *is* $\Omega(\min\{n, n^2/g^2\})$.

[2]As $\sqrt{n}/2$ is exactly the standard deviation of the number of coordinates that agree if we draw two vectors uniformly at random.

5.2.3 Applying Communication Complexity Lower Bounds in Distributed Computing

Communication complexity lower bounds are often used to obtain lower bounds in distributed computing. The classical reduction technique (see [101]) partitions the network into two parts A, B, with each player simulating the nodes in one part. Each players' input is used to construct that player's part of the network; for example, Alice's input may determine the edges connecting the nodes in A, or the input assignment to these nodes.

Since the players simulate disjoint parts of the network, and the structure and input of the part simulated by each player is determined only by that player's input, the two players can simulate their respective parts of the network with very little communication: the only information they need to exchange is the contents of messages crossing the cut (V_A, V_B). The partition is usually chosen so that the bandwidth going across the cut is small, call it b.

By simulating the algorithm until it halts and examining its behavior and output, the players are able to solve some communication complexity problem for which we have a lower bound, say $\Omega(c(n))$, on the number of bits necessary. If the algorithm requires $t(n)$ rounds to terminate, the simulation costs the players a total of $b \cdot t(n)$ bits, so we obtain a lower bound of $t(n) = \Omega(c(n)/b)$ on the time complexity of the algorithm. For example, this technique is used in [123] to obtain a lower bound on the complexity of computing the number of distinct elements in the input.

The reductions we give here are quite different in nature; we do not partition the network into two disjoint parts, and instead the players simulate potentially-overlapping sets of nodes. For instance, when we reduce from SET-DISJOINTNESS, each node of the network represents an element of the SET-DISJOINTNESS universe, and any node in the intersection of the inputs is simulated by both players. We still have a conceptual "information bottleneck" on which the bound relies, but it is no longer a sparse vertex cut in the graph. Since the two players simulate overlapping parts of the network, care must be taken to ensure that each player can simulate his or her nodes without needing to exchange too much information with the other player; this aspect of our reduction crucially relies on the fact that the network is directed.

Chapter 6

Data Aggregation in Directed Networks

In this chapter we study the complexity of computing basic functions in directed networks. We are particularly interested in the effect of *prior information* regarding parameters such as the network size and diameter, and in the difference between the deterministic and randomized complexity of various tasks.

We begin in Section 6.1 with a lower bound of $\Omega(n/B)$ rounds on computing an approximate count in networks of diameter 2, where B is the message size of the algorithm. This lower bound holds even if the diameter is known in advance to all participants. In Section 6.2 we give a lower bound of $\Omega(\sqrt{n/B})$ rounds on computing sensitive functions in networks of diameter 2, which holds only when it is *not* known in advance that the diameter is small. We conclude in Section 6.3 by giving a nearly-matching $\tilde{O}(\sqrt{n/B})$-round algorithm for solving the HF_n task, which allows us to compute simple duplication-insensitive functions.

6.1 Approximate and Exact Counting

We begin by describing a lower bound for ε-approximate counting or exact counting. In this setting we assume that nodes know some loose upper bound $N \geq n$ on the size of the network, and their goal is to determine the exact or approximate size. Since exact counting is a special case of approximate counting, we first describe the lower bound for approximate counting, and later discuss exact counting.

6.1.1 Lower Bound on Approximate Counting

Our lower bound on approximate counting is obtained by reduction from $\mathrm{GHD}_{N,\varepsilon N}$. Fix $\varepsilon \in (0,1)$, and suppose that we are given an ε-approximate counting algorithm $\mathcal{A}$. Given an instance (x, y) of $\mathrm{GHD}_{N,\varepsilon N}$, we construct a network $G_{x,y}$, in which Alice and Bob jointly simulate the execution of $\mathcal{A}$. When $\mathcal{A}$ terminates, Alice and Bob use the output of $\mathcal{A}$ to determine the correct answer to GHD on the instance (x, y). Since Alice knows only her input x and Bob knows only y, neither player knows the complete topology of the network $G_{x,y}$, which depends on both x and y. The players therefore cooperate to determine the topology of $G_{x,y}$ and simulate the execution of $\mathcal{A}$ in it.

Let $X, Y \subseteq [N]$ be the sets whose characteristic vectors are x and y, respectively. The network $G_{x,y}$ is given by $G_{x,y} = (V_{x,y}, E_{x,y})$, where $V_{x,y} = X \cup Y \cup \{a, b\}$ (for $a, b \notin [N]$), and $E_{x,y} = (\{a\} \times (V_{x,y} \setminus \{a\}) \cup (\{b\} \times (V_{x,y} \setminus \{b\})) \cup (X \times \{a\}) \cup (Y \times \{b\})$. See Fig. 6-1 for an illustration.

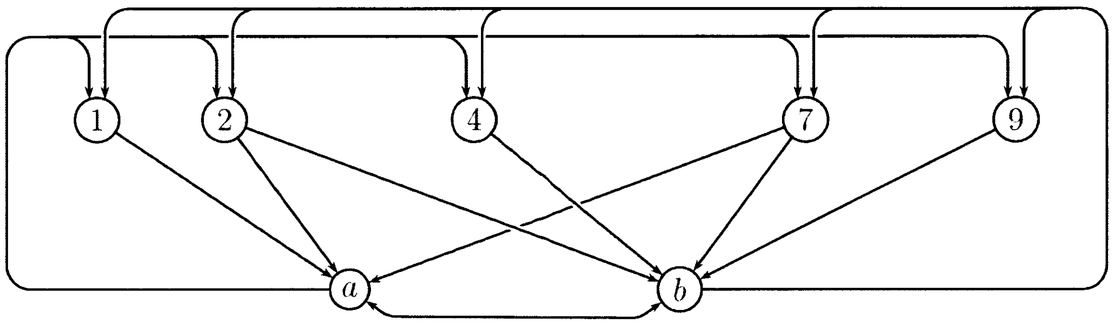


Figure 6-1: The network $G_{x,y}$ for inputs $x = 110000100, y = 010100101$ (that is, $X = \{1, 2, 7\}, Y = \{2, 4, 7, 9\}$).

The Hamming distance $\Delta(x, y)$ is closely related to the size of $G_{x,y}$:

Lemma 6.1. *For all $(x, y) \in (\{0,1\}^N)^2$, the graph $G_{x,y}$ is strongly connected, its diameter is 2, and its size is $|V_{x,y}| = |x| + |y| + \Delta(x, y))/2 + 2$.*

Proof. Since $\Delta(x, y) = |X \oplus Y|$, we have

$$\begin{aligned} \Delta(x, y) &= |(X \cup Y) \setminus (X \cap Y)| = |X \cup Y| - |X \cap Y| \\ &= |X \cup Y| - (|X| + |Y| - |X \cup Y|) = 2|X \cup Y| - |X| - |Y|, \end{aligned}$$

that is,

$$|X \cup Y| = (|X| + |Y| - \Delta(x, y))/2.$$

The size of $G_{x,y}$ is therefore given by

$$|V_{x,y}| = |X \cup Y| + 2 = (|x| + |y| - \Delta(x, y))/2.$$

The diameter of $G_{x,y}$ is 2, because nodes a and b have edges going in both directions, every node of $X \cup Y$ has an edge to either node a or node b (or both), and from nodes a, b there are edges to all other nodes. Strong connectivity also follows. □

Next we show that an efficient algorithm for approximating the size of diameter 2 networks leads to an efficient protocol for $\mathrm{GHD}_{N,\varepsilon N}$.

Lemma 6.2. *Given an ε-approximate counting algorithm $\mathcal{A}$ which outputs a correct answer after t rounds with probability at least $1 - \delta$ (where $\varepsilon, \delta \in (0, 1)$), one can* *construct a public-coin protocol for* $\mathrm{GHD}_{N,\varepsilon N}$ *which exchanges a total of* $O(Bt+\log N)$ *bits and succeeds with probability* $1-\delta$.

Proof. Given an instance (x,y), Alice and Bob simulate the execution of $\mathcal{A}$ in $G_{x,y}$ as follows. Alice locally simulates the nodes in $X \cup \{a\}$, and Bob locally simulates the nodes in $Y \cup \{b\}$. The shared random string is used to provide the randomness of all nodes in the network. (Since Alice and Bob do not initially know which of the nodes $\{1,\ldots,N\}$ are present, we interpret the shared random string as containing the randomness of each node $1,\ldots,N$ regardless of whether or not the node is in $X \cup Y$.) Notice that there can be some overlap, $X \cap Y$, which is simulated by both players independently.

The initial states of all nodes in $X \cup \{a\}$ and in $Y \cup \{b\}$ are known to Alice and Bob, respectively, because they depend only on the UIDs of these nodes and on the shared randomness. Each round of $\mathcal{A}$ is simulated as follows:

- Based on the states of their local simulations, Alice and Bob compute the messages sent by the nodes in $X \cup \{a\}$ and in $Y \cup \{b\}$, respectively.
- Alice sends to Bob the message sent by node a, and Bob sends to Alice the message sent by b. Following this exchange, Alice and Bob have all the messages received by each node they need to simulate.
- The players update the states of their local simulations by feeding to each node the messages it receives in $G_{x,y}$: the nodes of $X \cup Y$ receive the messages sent by a and b; node a receives the messages sent by nodes in $X \cup \{b\}$; and node b receives the messages sent by nodes in $Y \cup \{a\}$. (Note that Alice knows X and Bob knows Y, so the two players know which messages are supposed to be received by nodes a, b, respectively.)

Although Alice and Bob do not directly exchange information about the states of nodes in $X \cap Y$ — indeed, they do not *know* which nodes are in $X \cap Y$, and this is what makes the problem difficult — still their local simulations agree on the states of these nodes.

With probability at least $1-\delta$, after t rounds of the simulation node a halts and outputs an approximate count $\tilde{n}$ which satisfies $|\tilde{n}-n| \leq \varepsilon n$. When node a halts, Alice sends $\tilde{n}$ to Bob, and in addition Alice and Bob send each other $|X|$ and $|Y|$ (respectively). Let $\tilde{\Delta} = 2(\tilde{n}-2) - |X| - |Y|$. Both players output 0 if $\tilde{\Delta} < N/2$, and 1 if $\tilde{\Delta} \geq N/2$. (If node a fails to halt after t rounds, the players output an arbitrary answer.)

If $|\tilde{n}-n| \leq \varepsilon n$ then Lemma 6.1 shows that $|\tilde{\Delta} - \Delta(x,y)| = 2|\tilde{n}-n| \leq 2\varepsilon n \leq 2\varepsilon N$. Hence, with probability at least $1-\delta$, the players output the correct answer: if $\Delta(x,y) \geq N/2 + 2\varepsilon N$ then $\tilde{\Delta} \geq N/2$, and if $\Delta(x,y) < N/2 - 2\varepsilon N$ then $\tilde{\Delta} < N/2$.

The total number of bits sent during the protocol is $2Bt + 2\log(N)$. The communication complexity is therefore $O(Bt + \log N)$. □

From Theorem 5.9 and Lemma 6.2, we obtain the main lower bound of this section:

Theorem 6.3. *If* $B = \Omega(\log N)$, *then any randomized algorithm for computing an* ε*-approximate count requires* $\Omega((\min\{n, 1/\varepsilon^2\}/B)$ *rounds to succeed with probability 2/3 in networks of diameter 2.*

Proof. By Lemma 6.2, if there is a randomized distributed algorithm for counting with round complexity $t \geq 1$ and message size $B = O(\log N)$, then there is a randomized public-coin protocol for $\mathrm{GHD}_{N,\varepsilon N}$ with communication complexity $O(Bt + \log N) = O(Bt)$. By Theorem 5.2 we can transform the protocol into a private-coin protocol with communication complexity $O(Bt + \log N) = O(Bt)$. However, by Theorem 5.9, the public-coin communication complexity of $\mathrm{GHD}_{N,\varepsilon N}$ is $\Omega(\min\{N, 1/\varepsilon^2\} - \log N) = \Omega(\min\{n, 1/\varepsilon^2\})$. Therefore we must have $t = \Omega(\min\{n, 1/\varepsilon^2\}/B)$. □

Note that the lower bound is stated in terms of the actual size n, rather than the upper bound N on the size. Although our reduction in Lemma 6.2 is stated in terms of the upper bound N (we reduce from $\mathrm{GHD}_{N,\varepsilon N}$), the "hard" instances are the ones where n is roughly linear in N; it is always possible to solve GHD by exchanging the coordinates of indices i such that $x_i = 1$ or $y_i = 1$, and hence when $|X \cup Y| = n$ the problem can easily be solved in $O(n \log N)$ bits. It is therefore more informative to state our lower bound in terms of the actual size n of the network.

6.1.2 Deterministic and Exact Counting

The *deterministic* communication complexity of $\mathrm{GHD}_{N,g}$ is $\Omega(N)$ even when $g = c \cdot N$ for a sufficiently small constant c [29]; therefore we obtain the following lower bound for deterministic algorithms:

Theorem 6.4. *Deterministically computing an ε-approximate count for ε a sufficiently small constant requires $\Omega(n/B)$ rounds.*

Proof. As in Theorem 6.3, using the deterministic lower bound of $\Omega(N)$ on $\mathrm{GHD}_{N,\varepsilon N}$ instead of the randomized lower bound of $\Omega(\min\{N, 1/\varepsilon^2\})$. □

As for *exact* counting (deterministic or randomized), computing the exact count is as hard as computing a $(1/n)$-approximate count, so $\Omega(n/B)$ rounds are required.

Theorem 6.5. *Any randomized algorithm for computing the exact size of the network requires $\Omega(n/B)$ rounds to succeed with probability 2/3.*

Proof. This follows from Theorem 6.3 by plugging in $\varepsilon = 1/n$. □

6.1.3 A Nearly-Matching Upper Bound

The lower bound of Theorem 6.3 is nearly tight if the diameter of the network is known to be D.

Theorem 6.6. *There is a randomized ε-approximate counting for networks of diameter 2 that requires $\tilde{O}(\min\{n, 1/\varepsilon^2\}/B)$ rounds and succeeds with probability $1 - 1/\mathrm{poly}(N)$.*

Proof sketch. The case where $\min\{n, 1/\varepsilon^2\} = n$ is easy to handle: we use Algorithm 3.4 to disseminate the UIDs of all nodes using pipelining, with $B/\log n$ UIDs in each message. In $O(D + n \log n/B)$ rounds, all nodes acquire all UIDs, and can then compute the exact count. In particular, for $D = 2$, this matches Theorem 6.3 up to a $\log n$ factor.

For the case where $\min\{n, 1/\varepsilon^2\} = 1/\varepsilon^2$ we can use the same idea we used in Section 3.6 to compute an ε-approximate count. Each node computes an ℓ-tuple of exponential variables with rate 1, where $\ell = \Theta(\log N/\varepsilon^2)$, then rounds and truncates each variable according to (3.6.5). For D rounds, all nodes forward the smallest value heard so far for each coordinate of the tuple, and finally all nodes output the estimate $\hat{n}$ from (3.6.1).

As we saw in Section 3.6, with probability $1 - o(1)$ the estimate $\hat{n}_u$ computed by node u is an ε-approximation to the number of nodes that causally influence u; since in our case we know that the diameter is D, node u is causally influenced by all nodes after D iterations, so we obtain an ε-approximate count of the number of nodes in the network. The size of each ℓ-tuple is $O(\log N(\log\log N + \log(1/\varepsilon))/\varepsilon^2) = \tilde{O}(1/\varepsilon^2)$. Given bandwidth B, we can split every tuple into blocks of B bits each, obtaining an algorithm that runs in $\tilde{O}(D/(\varepsilon^2 B))$. This matches Theorem 6.3 up to a polylogarithmic factor for the case where $\min\{n, 1/\varepsilon^2\} = 1/\varepsilon^2$ and $D = 2$. □

There is some apparent slack between the upper bound and the lower bound: the lower bound of Theorem 6.3 applies to algorithms where each node individually succeeds with probability 2/3, while the upper bound sketched above has all nodes succeed together with probability $1 - 1/\text{poly}(N)$. This suggests that we can shave off a few log-factors by reducing the success probability to a constant, and eliminating the union bound that extends the analysis to all nodes rather than just one node. Indeed, if we set $\ell = \Theta(1/\varepsilon^2)$ instead of $\Theta(\log N/\varepsilon^2)$, we accomplish exactly that. However, there are some log-factors that are not so easily eliminated, and they come from rounding and truncating the exponential variables: the coarseness of our rounding scheme (see (3.6.5)) depends on ε, and the thresholds for truncation depend on both ε and $\log N$. It is conceivable that the dependence on $\log N$ can be eliminated by changing the truncation threshold so that it does not depend on N, and using a more sophisticated argument (in Theorem 3.35, where we use a union bound to show that with probability $1 - o(1)$ all nodes do not need to truncate their variables). But the dependence on ε, which contributes a $\log(1/\varepsilon)$ factor to the running time, seems inherent to this approach.

A similar problem has been studied in the context of communication complexity: given sets of size k, where $k = o(n)$, a naive solution to DISJ_n is for Alice to encode her input set using $O(k \log n)$ bits and send it to Bob. Can we eliminate the dependence on n, and use only $O(k)$ bits? It is known that for randomized algorithms the answer is yes [70], but for deterministic algorithms the answer is no [102]. Even in the randomized case, the solution is an elaborate protocol that uses public randomness (although for two players, public randomness almost comes for free [118]). It is interesting to ask whether these results carry over to the multi-player setting, which is closer to distributed computation. For example, do they carry over to multi-party promise Set Disjointness, where the players receive sets that either intersect at exactly one element or are pairwise disjoint? To our knowledge this question has not been investigated.

6.1.4 Lower Bound on Finding a Rooted Spanning Tree

We can use the reduction from Lemma 6.2 to show that finding a rooted spanning tree in directed networks is hard, even when the diameter of the network is known *a priori* to be 2 and the network graph admits no spanning trees of depth more than 3. This is because the network $G_{x,y}$ from Lemma 6.2 has exactly these properties. If one could find a rooted spanning tree of $G_{x,y}$ in t rounds, then an exact count could be computed in $t+3$ rounds by finding a rooted spanning tree and then "summing up the tree" (convergecast). Since exact counting requires $\Omega(n/B)$ rounds, finding a rooted spanning tree must also require $\Omega(n/B)$ rounds.

This lower bound was included in [96], where the results in this chapter first appeared. However, it is not a very satisfying lower bound, because it depends on the nodes not knowing the count n in advance; if they do, we cannot argue that finding the count requires $\Omega(n/B)$ rounds. This seems like "the wrong reason" for this hardness result, as knowing the size of the network does not appear to help in any way. And indeed, in Chapter 7 we will introduce a new multi-player problem, *task allocation*, and through it obtain an $\Omega(n/B)$-round lower bound on finding rooted spanning trees in networks whose size is known to be n.

6.2 Lower Bounds on Sensitive Functions

In this section we study the complexity of computing sensitive functions, such as the minimum or maximum input value. In contrast to the previous section, here we are interested in instances where the diameter of the network is not known *a priori* to be small, but the algorithm is deployed in a network that *does* in practice have a small diameter. We ask to what degree an algorithm can exploit the small diameter of the network, and in particular, how easy is it to detect that the network has a small diameter.

In Section 2.2 we defined two notions of sensitivity that apply regardless of whether the number of nodes is known in advance:

- *Globally-sensitive functions*: For each n, there exists an input assignment (multiset) X of size n in the domain of f, such that changing any single input in X changes the value of f.
- *ε-sensitive functions* (for $\varepsilon \in (0,1)$): For each n, there exists an input assignment X of size n in the domain of f, such changing any $\varepsilon \cdot n$ inputs in X changes the value of f.

It is easy to see that these two notions are closely related to the problem of hearing from everyone (HF_n) and hearing from a $(1-\varepsilon)$-fraction of nodes ($HF_{(1-\varepsilon)n}$):

Proposition 6.7. *Let $\mathcal{A}$ be an algorithm that computes a globally-sensitive function f. Then for any network size n, there is an input assignment X such that when $\mathcal{A}$ is executed with input X in any graph of size n, no node halts before hearing from everyone (i.e., $\mathcal{A}$ with input X solves HF_n). Similarly, if $\mathcal{A}'$ computes an ε-sensitive* *function f', then for any network size n there is an input X' such that $\mathcal{A}'$ with input X' solves $HF_{(1-\varepsilon)n}$.*

Proof. Let X be an input assignment on which f is globally-sensitive. Suppose for the sake of contradiction that in some graph G, when $\mathcal{A}$ is executed with input X, HF_n is not solved: some node u halts at time t such that $\Upsilon^t(x) \neq V$. Then there is some node $v \notin \Upsilon^t(x)$, and we can change node v's input to obtain a new input assignment X' differing from X by exactly one element. Since f is globally-sensitive on X, we have $f(X) \neq f(X')$. But on the other hand, Proposition 2.28 shows that u cannot distinguish the execution with input X from the execution with input X', so its output is the same in both.

For ε-sensitive functions the proof is similar: if some node u halts before hearing from $(1-\varepsilon)n$ nodes, then there exist at least $\varepsilon \cdot n$ nodes from which u does not hear. We can change the input to these nodes and obtain an assignment on which the value of f' differs from $f'(X')$, but u does not notice this change. □

It follows from Proposition 6.7 that a lower bound on HF_n yields a lower bound on computing any globally-sensitive function, and a lower bound on $HF_{(1-\varepsilon)n}$ yields a lower bound on computing any ε-sensitive function. In the current section we prove two such lower bounds, one for randomized algorithms solving HF_n and one for deterministic algorithms solving $HF_{(1-\varepsilon)n}$. In Section 6.3 we will give a nearly-optimal randomized algorithm for solving HF_n. This algorithm by itself is not sufficient to compute any function, but by Proposition 3.8 it is sufficient to compute duplication-insensitive single-valued functions over at most $O(\log n)$ input values, because HF_n is $(0, b)$-complete for functions over b input values.

6.2.1 Lower Bounds on the HF_m Task

We now show that distinguishing networks of diameter 2 from networks of diameter $\Omega(\sqrt{n/B})$ requires $\Omega(\sqrt{n/B})$ rounds in the worst case. More formally, we show that when the size of the network is known, the UID space is $1, \ldots, n$, and no *a priori* bound on the diameter is known,

(a) Any randomized algorithm for HF_n requires $\Omega(\sqrt{n/B})$ rounds to succeed with constant probability, even when executed in a network of diameter 2; and

(b) For any $\varepsilon \in (0, 1/2)$, any deterministic algorithm for $HF_{(1-\varepsilon)n}$ requires $\Omega(\sqrt{n/B})$ rounds, again when executed in networks of diameter 2.

(Of course, in networks of diameter 2, two rounds are sufficient to solve HF_n, if only the algorithm *knew* that the diameter is 2.)

Fix an algorithm $\mathcal{A}$ for HF_m and a network size $n \geq m$. We describe a reduction from Set Disjointness to HF_n, which we will use to show the hardness of HF_n for randomized algorithms. We then use the same reduction to show the hardness of $HF_{(1-\varepsilon)n}$ for deterministic algorithms, reducing this time from Gap Set Disjointness (which is merely Set Disjointness with a gap promise).

6.2.2 A Reduction from Set Disjointness to HF_m

Our reduction is parameterized by m, the number of nodes from which each node must hear in HF_m, and by a specific algorithm $\mathcal{A}$. As in Section 6.1, in the reduction we construct a network G based on the instance of Set Disjointness given to Alice and Bob. The two players then simulate the execution of $\mathcal{A}$ in G, and output an answer to Set Disjointness (or Gap Set Disjointness) based on the behavior of $\mathcal{A}$ in G — in particular, based on the time when $\mathcal{A}$ terminates. We now describe the construction of the network and the simulation used by Alice and Bob.

The construction has several parameters. First, let $t_{\mathcal{A}}$ be a bound on the time complexity of $\mathcal{A}$ in networks of $n+2$ nodes with diameter 2: specifically, when $\mathcal{A}$ is executed in any network of size n with node UIDs $1, \ldots, n', a, b$, where $n' = n-2$, with probability at least $2/3$ all nodes halt by time $t_{\mathcal{A}}$. Based on $t_{\mathcal{A}}$ and on the parameter m, we choose a *segment length* $s \geq t_{\mathcal{A}} + 1$ which will be fixed later. Informally, in the reduction nodes must distinguish diameter 2 networks from diameter $s+2$, and we will show that this requires $\Omega(n/s)$ rounds in the worst-case.

Assume for simplicity that s divides n'. We divide the nodes $1, \ldots, n'$ into *segments* $S_1, \ldots, S_{n'/s}$, each of size s, where $S_i := \{(i-1) \cdot s + 1, (i-1) \cdot s + 2, \ldots, i \cdot s\}$. Each segment S_i is further subdivided into two parts: a *back end* S_i^B containing nodes $(i-1) \cdot s + 1, \ldots, i \cdot s - t_{\mathcal{A}}$, and a *front-end* S_i^F containing the remaining nodes, $i \cdot s - t_{\mathcal{A}} + 1, \ldots, i \cdot s$. In the sequel we implicitly use wrap-around for node indices, so that $-1 \equiv n'$, $-2 \equiv n' - 1$, and so on.

We are now ready to describe the reduction itself. The reduction is from $\text{DISJ}_{n'/s}$, that is, Set Disjointness (or Gap Set Disjointness) with a universe of n'/s elements; each segment S_i represents a single element of the universe. Given an instance (x, y) of $\text{DISJ}_{n'/s}$, we define a network $G_{s,x,y} := (\{1, \ldots, n', a, b\}, E_{s,x,y})$, where

- Nodes a, b have edges to all nodes of the graph.
- Nodes $1, \ldots, n$ are connected in a directed cycle: for each $i \in [n']$ we have $(i, i+1) \in E_{s,x,y}$.
- In each segment S_i, the last node (node $i \cdot s$) is connected to node a. (This is to ensure strong connectivity and a bound of $s+2$ on the diameter.)
- For all $i \notin X$ and for all $v \in S_i$ we have $(v, a) \in E_{s,x,y}$; similarly, for all $i \notin Y$ and for all $v \in S_i$ we have $(v, b) \in E_{s,x,y}$.

Here, X and Y are the sets whose characteristic vectors are x, y respectively. See Fig. 6-2 for an illustration of $G_{s,x,y}$.

Let us describe informally why $G_{s,x,y}$ "captures" the $\text{DISJ}_{n'/s}$ problem. With the exception of the last node in each segment (which is always connected to node a), the nodes in segment S_i are connected to node a iff Alice did *not* receive i in her input, and connected to node b iff Bob did not receive i in his input. Therefore, if there exists an element i in the intersection $X \cap Y = \overline{\overline{X} \cup \overline{Y}}$, the nodes of the corresponding segment S_i, with the exception of the last node, will not be connected to either node a or node b. These nodes are only connected to the rest of the graph by the cycle

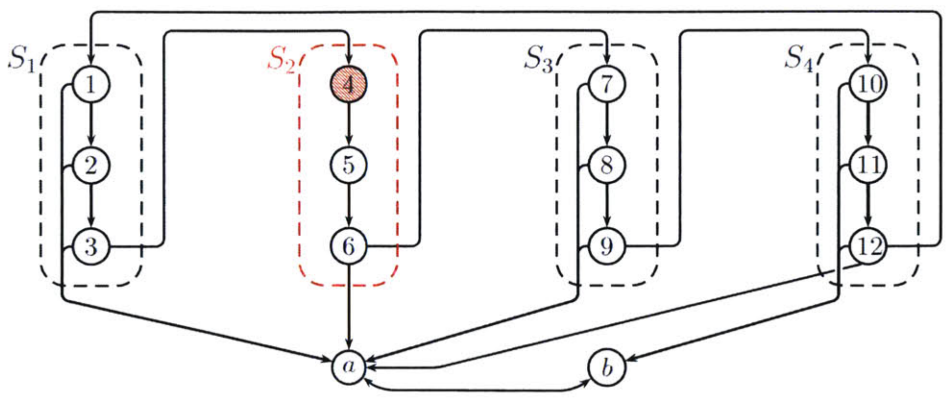


Figure 6-2: The network $G_{s,x,y}$ from Theorem 6.10, with $n = 12, t_\mathcal{A} = 2, s = t_\mathcal{A} + 1 = 3$. Edges from a, b to nodes $1, \ldots, 12$ are omitted for clarity. The DISJ_4 instance shown here is $X = \{2, 4\}, Y = \{1, 2, 3\}$. Since $2 \in X \cap Y$, all S_2 nodes except the last (node 6) are not connected to a or to b. Therefore $4 \notin \Upsilon^{t_\mathcal{A}}(a)$, i.e., two rounds are not sufficient for node a to hear from node 4.

edges $(i-1) \cdot s + 1 \to (i-1) \cdot s + 2 \to \ldots \to i \cdot s$. Consequently the diameter of the graph is $s + 1 > t_\mathcal{A}$ in this case. In $t_\mathcal{A}$ rounds, nodes a and b can only hear from the last $t_\mathcal{A}$ nodes of segment S_i, i.e., only from the front-end S_i^F; for each segment S_i such that $i \in X \cap Y$, $|S_i^B| = s - t_\mathcal{A}$ nodes are missing from $\Upsilon^{t_\mathcal{A}}(a)$.

On the other hand, if $X \cap Y = \emptyset$ (or equivalently, $\overline{X} \cup \overline{Y} = \{1, \ldots, n'/s\}$), all nodes in all segments are connected to either node a or node b, and the diameter of the graph is 2.

More formally, we observe that the graph $G_{s,x,y}$ has the following characteristics.

Lemma 6.8. *For any $x, y \in \{0, 1\}^{n'}$,*

(a) The graph $G_{s,x,y}$ is strongly connected,

(b) For all $i \in X \cap Y$ and for all $v \in S_i^B$ we have $v \notin \Upsilon^{t_\mathcal{A}}(a)$ and $v \notin \Upsilon^{t_\mathcal{A}}(b)$,

(c) If $X \cap Y = \emptyset$, the diameter of $G_{s,x,y}$ is 2, and

(d) $|\Upsilon^{t_\mathcal{A}}(a) \cap \Upsilon^{t_\mathcal{A}}(b)| \leq n' - |X \cap Y| \cdot (s - t_\mathcal{A})$ (and similarly for b).

Proof. From each node $(i-1) \cdot s + j \in S_i$ (where $1 \leq i \leq n'/s$, $1 \leq j \leq s$), the distance to nodes a, b is exactly $(s - j) + 1$ if $i \in X \cap Y$, and the distance is either 1 or 2 if $i \notin X \cap Y$. Since a and b have edges to all other nodes, $G_{s,x,y}$ is strongly-connected. Also, if $i \in X \cap Y$ then $i \in X$ and $i \in Y$, so the distance from node $(i-1) \cdot s + j$ to nodes a, b is at least $(s - j) + 1$; if this is a back-end node, then $j \leq s - t_\mathcal{A}$, so the distance exceeds $s - j \geq t_\mathcal{A}$. This shows part (b) of the lemma. Part (c) follows from the fact that if $X \cap Y = \emptyset$ then $\overline{X} \cup \overline{Y} = \{1, \ldots, n'\}$, and therefore each node is connected directly either to node a or to node b or both; from these nodes to all other nodes the distance is 1. And finally, part (d) follows from part (b), which shows that all back-end nodes of all segments $i \in X \cap Y$ are *not* in $\Upsilon^{t_{\mathcal{A}}}(a)$; there are $s - t_{\mathcal{A}}$ nodes in each segment's back-end, so part (d) follows. □

Alice and Bob simulate the execution of $\mathcal{A}$ in $G_{s,x,y}$ in a slightly different manner than in Lemma 6.2; here both players simulate nodes $1, \ldots, n'$ regardless of the input instance, and in addition Alice simulates node a and Bob simulates node b. The remainder of the simulation is the same as in Lemma 6.2.

Proposition 6.9. *Given inputs x and y respectively, and a shared string representing the randomness of all nodes, Alice and Bob can each simulate nodes $\{a, 1, \ldots, n'\}$ and $\{b, 1, \ldots, n'\}$ (respectively) throughout rounds $1, \ldots, t_{\mathcal{A}}$ of the execution of $\mathcal{A}$ in $G_{s,x,y}$. The simulation requires $2B$ bits per simulated round.*

Proof. As in Lemma 6.2, the players store a local copy of the states of all nodes they need to simulate, and exchange only the messages sent by nodes a and b (simulated by Alice and Bob, respectively). Each node $1, \ldots, n$ receives the messages sent by a and b, which are known to both players after their exchange. The players are also able to compute the message generated by each node $1, \ldots, n'$, and moreover, Alice knows which incoming edges node a has (they depend only on X) and Bob knows which incoming edges node b has (they depend only on Y). Therefore the players can compute the next state of each node they need to simulate. □

It remains only to put the pieces together to obtain the following lower bounds.

Theorem 6.10. *If the diameter of the network is not known initially, any randomized algorithm for solving HF_n requires $\Omega(\sqrt{n/B})$ rounds with probability at least $2/3$ when executed in networks of diameter 2.*

Proof. Fix an algorithm $\mathcal{A}$, and let $t_{\mathcal{A}}$ be defined as above. Fix a segment length of $s := t_{\mathcal{A}} + 1$ (so that the back-end of each segment contains exactly one node).

Given an instance (x, y) of $\text{DISJ}_{n'/s}$, where $n' = n - 2$, Alice and Bob jointly simulate the first $t_{\mathcal{A}}$ rounds in the execution of $\mathcal{A}$ in $G_{s,x,y}$ as in Proposition 6.9. After $t_{\mathcal{A}}$ rounds, Alice informs Bob whether or not node a has halted in the simulation. If node a has halted, the players output "$X \cap Y = \emptyset$"; otherwise they output "$X \cap Y \neq \emptyset$".

As we saw in Lemma 6.8, if $X \cap Y = \emptyset$ then the diameter of $G_{t_{\mathcal{A}}+1,x,y}$ is 2, so with probability at least 2/3 all nodes halt after $t_{\mathcal{A}}$ rounds and Alice and Bob output "$X \cap Y = \emptyset$". On the other hand, if $X \cap Y \neq \emptyset$, then by time $t_{\mathcal{A}}$ node a has not heard from all nodes, as Lemma 6.8 shows that at least $(s - t_{\mathcal{A}}) \cdot |X \cap Y| = |X \cap Y| > 0$ nodes are missing from $\Upsilon^{t_{\mathcal{A}}}(a)$. Consequently, with probability at least 2/3, node a does not halt by time $t_{\mathcal{A}}$ and the players output "$X \cap Y \neq \emptyset$".

The total number of bits exchanged by the players in the protocol above is $2B \cdot t_{\mathcal{A}} + 1$, because Alice and Bob only send each other the messages output by nodes a and b, plus one bit needed for Alice to inform Bob whether node a has halted. An additional $O(\log(n/t_{\mathcal{A}}))$ bits are required to obtain a private-coin protocol. Since the randomized communication complexity of $\text{DISJ}_{\lfloor n'/(t_{\mathcal{A}}+1) \rfloor}$ is $\Omega(n'/t_{\mathcal{A}}) = \Omega(n/t_{\mathcal{A}})$, we must have $2B \cdot t_{\mathcal{A}} + 1 = \Omega(n/t_{\mathcal{A}})$, or in other words, $t_{\mathcal{A}} = \Omega(\sqrt{n/B})$. □

Theorem 6.11. *If the diameter of the network is not known initially, then for any $\varepsilon \in (0, 1/2)$, any deterministic algorithm for solving $HF_{(1-\varepsilon)n}$ requires $\Omega(\sqrt{n/B})$ rounds, even in networks of diameter 2.*

Proof. The proof is similar to that of Theorem 6.10, except that we now reduce from GAP-DISJ$_{\lfloor n'/s \rfloor, \varepsilon' \lfloor n'/s \rfloor}$ for an appropriately chosen constant $\varepsilon' \in (0, 1/2)$, and the segment length s is also chosen differently.

Fix a deterministic algorithm $\mathcal{A}$ for $HF_{(1-\varepsilon)n}$, and let $t_{\mathcal{A}} = t_{\mathcal{A}}(n)$ be the maximal time at which the algorithm halts in any network of diameter 2 and size n. Assume that $t_{\mathcal{A}} = o(n)$, otherwise we are done. We must now choose a segment length $s = \Theta(t_{\mathcal{A}})$ so that the following conditions hold:

(a) If $X \cap Y = \emptyset$, then the diameter of $G_{s,x,y}$ is 2. This ensures that in "yes" instances, all nodes halt by time $t_{\mathcal{A}}$.

(b) If $|X \cap Y| \geq \varepsilon' \lfloor n/s \rfloor$ then we have $|\Upsilon^{t_{\mathcal{A}}}(a)| < (1-\varepsilon)n = (1-\varepsilon)(n'+2)$. This ensures that in "no" instances, node a *cannot* halt by time $t_{\mathcal{A}}$, because it has not heard from sufficiently many nodes.

Whenever these conditions hold, the protocol from Theorem 6.10 allows Alice and Bob to solve GAP-DISJ$_{\lfloor n/s \rfloor, \varepsilon' \lfloor n/s \rfloor}$ as well.

From Lemma 6.8 we see that condition (a) holds regardless of our choice of s. As for condition (b), from part (d) of Lemma 6.8, it is sufficient to choose $s := \alpha t_{\mathcal{A}}$ and ε' so that

$$n' - \varepsilon' \left\lfloor \frac{n'}{\alpha t_{\mathcal{A}}} \right\rfloor \cdot (\alpha - 1) t_{\mathcal{A}} < (1-\varepsilon) n', \tag{6.2.1}$$

that is,

$$\varepsilon' \left\lfloor \frac{n'}{\alpha t_{\mathcal{A}}} \right\rfloor \cdot (\alpha - 1) t_{\mathcal{A}} > \varepsilon n'. \tag{6.2.2}$$

Fix some constants $\varepsilon' \in (\varepsilon, 1/2)$ and α such that $(\alpha - 1)/\alpha > \varepsilon/\varepsilon'$. (There exist such ε', α, because $\varepsilon \in (0, 1/2)$ and $\lim_{\alpha \to \infty} (\alpha - 1)/\alpha = 1 > \varepsilon/\varepsilon'$.) Since we assumed that $t_{\mathcal{A}} = o(n)$ (and hence $n'/t_{\mathcal{A}} = \omega(1)$), and since $(\alpha - 1)/\alpha > \varepsilon/\varepsilon'$, there exists an n_0 such that for all $n \geq n_0$ we have

$$\left\lfloor \frac{n'}{\alpha t_{\mathcal{A}}} \right\rfloor \cdot (\alpha - 1) t_{\mathcal{A}} > \frac{\varepsilon}{\varepsilon'} n'. \tag{6.2.3}$$

Thus, (6.2.2) is satisfied for all $n \geq n_0$. For this choice of $s = \alpha t_{\mathcal{A}}, \varepsilon'$ and for all $n \geq n_0$, the reduction from Theorem 6.10 yields a protocol with communication complexity $2Bt_{\mathcal{A}} + 1$ for GAP-DISJ$_{n'', \varepsilon' n''}$, where $n'' = \lfloor n'/s \rfloor = O(n/t_{\mathcal{A}})$. Because GAP-DISJ is linearly hard for deterministic protocols even when the gap is linear in the universe size (Theorem 5.6), we must have $2Bt_{\mathcal{A}} + 1 = \Omega(n/t_{\mathcal{A}})$, i.e., $t_{\mathcal{A}} = \Omega(\sqrt{n/B})$. □

6.2.3 Discussion and Extensions

The construction in this section can be modified to show a few related results.

In Theorems 6.10 and 6.11 we assumed that no upper bound on the diameter of the network is known in advance. Suppose now that some upper bound $\bar{D}$ on the diameter is known in advance. We can show:

Theorem 6.12. *Any randomized algorithm for computing a globally-sensitive function, requires* $\Omega(\min\left\{\bar{D}, \sqrt{n/B}\right\})$ *rounds when executed in networks of diameter 2. So does any deterministic algorithm for computing an* ε*-sensitive function for* $\varepsilon \in (0, 1/2)$.

Proof. To see this, observe that the diameter of the network $G_{s,x,y}$ never exceeds $s+2$: if $X \cap Y = \emptyset$ then the diameter of $G_{s,x,y}$ is 2, and if $X \cap Y \neq \emptyset$ then the diameter is $s+2$. If $\bar{D} = \Omega(\sqrt{n/B})$, knowing an upper bound of $\bar{D}$ is "not useful": with an easy modification of constants, the proof of Theorem 6.10 goes through, because it constructs a network of diameter at most $s + 2 = O(\sqrt{n/B}) = O(\bar{D})$. Therefore in this case the running time is still $\Omega(\sqrt{n/B})$. On the other hand, suppose that $\bar{D} = o(\sqrt{n/B})$. We will show that in this case the running time is $\Omega(\bar{D})$.

To that end, suppose that $\bar{D} = o(\sqrt{n/B})$ and we are given an HF_n-algorithm (or similarly, a deterministic $HF_{(1-\varepsilon)n}$-algorithm) $\mathcal{A}$ with $t_{\mathcal{A}} < \bar{D} - 2$. If we use a segment length of $s = t_{\mathcal{A}} + 1 < \bar{D}$ in Theorem 6.10, the diameter upper bound of $\mathcal{D}$ is not violated in $G_{s,x,y}$. For this choice of s, the reduction from Theorem 6.10 allows us to solve $\text{DISJ}_{\lfloor n/s \rfloor}$, where $\lfloor n/s \rfloor \geq \lfloor n/(\bar{D}-2) \rfloor$, using less than $2(\bar{D}-2)B+1$ bits. We must have $2(\bar{D}-2)B+1 = \Omega(n/\bar{D})$, that is, $\bar{D} = \Omega(\sqrt{n/B})$, contradicting our assumption that $\bar{D} = o(\sqrt{n/B})$. □

Next, consider the problem of finding an approximate count when the diameter of the network is not known in advance. In Section 6.1 we gave a lower bound that assumes the diameter is known to be 2; however, it is possible to obtain a stronger lower bound for the case where the diameter is unknown, by following the lines of Lemma 3.6. Let N be the best upper bound known in advance on the count. Then we have the following lower bound:

Theorem 6.13. *If the diameter of the network is not known in advance, then any randomized algorithm requires* $\Omega(\sqrt{n/B})$ *rounds to distinguish a network of size* n *from a network of size* N *with probability* $2/3$*, even when executed in networks of diameter 2.*

Proof. We will show that in order to distinguish a network of size n from a network of size N, nodes a, b must solve a Set Disjointness instance of size $O(n/t_{\mathcal{A}})$, so that as in Theorem 6.10, $t_{\mathcal{A}} = \Omega(\sqrt{n/B})$ rounds are required.

Recall that in $G_{s,x,y}$, the distance from any node $(i-1)\cdot s+1$ where $i \in X \cap Y$ to nodes a and b is $s > t_{\mathcal{A}}$. Thus, when $X \cap Y \neq \emptyset$, we can choose a node $v := (i-1)\cdot s+1$ where $i \in X \cap Y$, and "hide" nodes $n+1, \ldots, N$ behind it, adding edges from nodes $n+1, \ldots, N$ to v and from nodes a, b to nodes $n+1, \ldots, N$. Let $G'_{s,x,y}$ be the resulting network. Since the distance from node v to nodes a, b exceeds $t_{\mathcal{A}}$, and the new nodes $n+1, \ldots, N$ are connected only to node v, $t_{\mathcal{A}}$ rounds are insufficient for nodes a, b to distinguish $G_{s,x,y}$ from $G'_{s,x,y}$. Therefore, if $X \cap Y \neq \emptyset$, an algorithm for distinguishing networks of size $n+2$ from networks of size N cannot terminate by time $t_{\mathcal{A}}$ in $G_{s,x,y}$ (except with small probability). This is sufficient to carry out the reduction from Theorem 6.10 exactly as before, obtaining an $\Omega(\sqrt{n/B})$ lower bound on any non-trivial approximation of the count. □

6.3 A Nearly-Optimal Algorithm for HF_n

In the previous section we showed that the HF_n problem requires $\Omega(\sqrt{n/B})$ rounds, even when the diameter of the network is 2. We now give a nearly-matching algorithm, which solves HF_n in $2D + \tilde{O}(\sqrt{n/B})$ rounds in networks of diameter D, where D is not known in advance.

6.3.1 Discussion and High-Level Sketch of the Algorithm

Let us first discuss the connection between HF_n and approximate counting. In Section 6.1 we showed that obtaining an ε-approximate count of an n-node network requires $\Omega(\min\{n, 1/\varepsilon^2\}/B)$ rounds. On first impression, this might seem to imply that our $\Omega(\sqrt{n/B})$ lower bound on HF_n (Theorem 6.10) is not tight: to solve HF_n it appears we need to count how many nodes we have heard from to a precision of better than ± 1, so that we do not accidentally halt before hearing from some node. This means we must set $\varepsilon < 1/n$, so we will need $\Omega(n/B)$ rounds to count. However, a more careful examination reveals that we can get away with a much coarser approximate count, by counting the *new* nodes we have heard from recently rather than the total number of nodes. From strong connectivity, if node u has *not* heard from all nodes at time t, then we have $|\Upsilon^t(u) \setminus \Upsilon^{t'}| \geq t - t'$ for all $t' \leq t$. If we could estimate $|\Upsilon^t(u) \setminus \Upsilon^{t'}(u)|$ to a *constant* accuracy — that is, with a additive error of $\pm\varepsilon(t-t')$ where $\varepsilon \in (0,1)$ is some constant — then we could accurately test whether $\Upsilon^t(u) \setminus \Upsilon^{t'}(u)$ is empty or not. If we determine that $\Upsilon^t(u) \setminus \Upsilon^{t'}(u)$ is empty, then node u may halt at time t; and if node u has heard from everyone at time t', then $\Upsilon^t(u) \setminus \Upsilon^{t'}(u)$ will be empty. Thus we can solve HF_n in time roughly $D + t - t'$.

Unfortunately, the approximate counting scheme from Section 3.6 does not lend itself to estimating the size of the set difference $\Upsilon^t(u) \setminus \Upsilon^{t'}(u)$. One might try to estimate the sizes of $\Upsilon^t(u)$ and $\Upsilon^{t'}(u)$ separately using the approach from 3.6 and taking the difference: this corresponds to measuring the *decrease in the value of* $\tilde{n}_u$ between time t' and t (where $\tilde{n}_u$ is the estimate from 3.6). However, if $\Upsilon^{t'}(u)$ is large (e.g., almost n), the absolute error of $\tilde{n}_u(\Upsilon^{t'}(u))$ may be large enough to overwhelm the true difference in sizes, $|\Upsilon^t(u) \setminus \Upsilon^{t'}(u)|$, causing $\tilde{n}_u(\Upsilon^t(u)) - \tilde{n}_u(\Upsilon^{t'}(u))$ to be zero even when $|\Upsilon^t(u) \setminus \Upsilon^{t'}(u)| \geq t - t'$.

Instead, we use the following high-level idea. Initially, each node selects itself with probability $1/\sqrt{n}$. The UIDs of the selected nodes are disseminated by pipelining (see Section 3.4.2). Each node waits until there is some interval of $2\sqrt{n}$ contiguous rounds in which it does not receive any new UID, and then halts.

The expected number of nodes that select themselves is $\sqrt{n}$, and the latency of pipelining is exactly the number of tokens, so the expected running time is $D + 3\sqrt{n}$: after $D + \sqrt{n}$ rounds all tokens are received, and after waiting an additional $2\sqrt{n}$ rounds, all nodes halt. Also, if node u halts at time t, then no new UIDs were received at u between time t and time $t - 2\sqrt{n}$. Notice that at time $t - 2\sqrt{n}$, node u has not received any UIDs of nodes at in-distance greater than $t - 2\sqrt{n}$ from u, because there is not enough time for these UIDs to reach u; and on the other hand, at time t, "in expectation", node u has already received all the UIDs from distance $t - \sqrt{n}$. It follows that all UIDs from a distance no smaller than $t - 2\sqrt{n}$ but no greater than $t - \sqrt{n}$ are received between time $t - 2\sqrt{n}$ and time t. By strong connectivity, unless $\Upsilon^t(u) = V$, there are at least $\sqrt{n}$ nodes whose distances from u are in this range; in expectation at least one of these nodes selects itself, and its UID is received between time $t - 2\sqrt{n}$ and time t, preventing node u from halting at time t.

To turn this idea into a concrete algorithm, we must overcome three hurdles:

- First, and easiest to overcome, is the fact that our algorithm should succeed with high probability rather than just "in expectation".

- We do not wish to assume that the nodes initially know the size n of the network.[1] This means that nodes do not know the appropriate probability to select themselves $(1/\sqrt{n})$, nor do they know how long to wait for new tokens $(2\sqrt{n})$. To deal with this challenge we will use exponentially-increasing guesses for the size of the network, but we will see that adapting the idea above to a guess instead of the true size n is slightly more complicated than one might expect.

- Finally, note that the correctness and latency guarantee of pipelining depends on all nodes participating until all tokens are disseminated everywhere. In our sketch above, we allowed nodes to simply halt when they have not received any new UIDs for sufficiently long; however, if nodes actually do this, their lack of further participation may effectively cause a partition in the network and foil the other nodes.

  To address this, we add a *termination protocol*, which all nodes execute alongside the *estimation protocol* sketched above. The estimation protocol is responsible for identifying a time t such that $\Upsilon^t(u) = V$; once such a time has arrived, we say that node u is *ready*. However, node u continues participating in pipelining until it is certain that all other nodes are also ready, and only then halts.

Let us now turn to the technical details.

6.3.2 Token Selection Schemes

Our algorithm is based on having nodes generate some initial set of tokens, and then pipeline these tokens and halt when no new token is received "for a long time". Let us use $A(u)$ to denote the set of tokens generated by node u, and $A(S) = \bigcup_{u \in S} A(u)$.

[1] We did assume this for our lower bound, to make it stronger, but it is not an appealing assumption on which to base an algorithm for HF_n.

A *token selection scheme* is defined as follows. For each guess k, for the logarithm of the size of the network, $k = \log\log N, \dots, \log N$, the scheme specifies a parameter ℓ_k and a mechanism by which each node selects a set of tokens from the set $\{1, \dots, \ell_k\}$. We define $L_k := \sum_{i \le k} \ell_k$. Assume that β tokens can fit in a B-bit message, and let $\tau_k := \lceil L_k / B \rceil$.

For each guess k, let $A_k(S)$ denote the set of tokens generated by the nodes of $S \subseteq V$, and let T_k be the total number of tokens generated by all nodes. Also, let

$$\hat{k} := \max\{\log\log N, \log n\}$$

be the "correct" value of the guess k when executed in networks of size n. The token selection scheme must guarantee that the following event G occurs with high probability:

(G1) $T_{\hat{k}} \le \ell_{\hat{k}}/2$, that is, on level $\hat{k}$, no more than $\ell_{\hat{k}}/2$ tokens are generated by all nodes together.

(G2) For any node $u \in V$, level $k \le \hat{k}$ and distance $d \ge \tau_k$ such that $\Upsilon^d(u) \subset V$, either $|A_k(\Upsilon^{d-\tau_k}(u))| > \ell_k/2$ or $A_k(\Upsilon^d(u)) \setminus A_k(\Upsilon^{d-\tau_k}(u)) \ne \emptyset$.

We will see in Section 6.3.3 that given a token selection scheme satisfying these requirements, we can solve HF_n in $O(D + \ell_{\hat{k}})$ rounds with high probability. Thus, our goal is to find a scheme satisfying these requirements and minimizing the parameter $\ell_{\hat{k}}$ (to the extent possible in light of our lower bound from Section 6.2).

In Section 6.3.1 we sketched a scheme where each node selects its own UID with probability $1/\sqrt{n}$, and this is the only token it generates. When we attempt to adapt this to a guess k instead of the true size n, the immediate inclination is to have each node select itself with probability $1/\sqrt{k}$. This preserves the essential property that in each $\sqrt{k}$-neighborhood, in expectation at least one node selects itself. Unfortunately, the other side of the equation is that we must not have too *many* nodes select themselves, as we must still collect all tokens from the $\sqrt{k}$-neighborhood using pipelining. If each node selects itself with probability $1/\sqrt{k}$, we will have $n/\sqrt{k}$ tokens in expectation, so the pipelining latency will be $n/\sqrt{k}$. If $k < n$, then $2\sqrt{k}$ rounds will not be sufficient to collect all tokens, and we may mistakenly think that no nodes in the neighborhood selected themselves when in fact some nodes did.

To restrict the total number of tokens, we simply have nodes select not their own UID but a random name from the domain $\{1, \dots, \ell_k\}$, where

$$\ell_k := 12(c+3)\sqrt{2^{k+4}\beta \ln N} + 1.$$

Here c is a constant corresponding to the desired success probability $(1 - 1/N^c)$. Note that since our guesses satisfy $k \ge \log\log N$, we have $2^{k+4} \ge 16 \log N > \ln N$, and therefore $\ell_k \ge 12(c+3)\ln N + 1$ for each guess k that we consider. (We do not attempt to optimize constants here.)

After choosing a random name, each node selects its name with probability

$$p_k := \frac{\ell_k}{2^{k+2}}.$$

Let us now show that the property G is satisfied with high probability.

Lemma 6.14. *For any subset $S \subseteq V$ of size at least τ_k, with probability at least $1 - 1/(2N^{c+3})$ we have either $|A_k(V \setminus S)| > \ell_k/2$ or $A_k(S) \setminus A_k(V \setminus S) \neq \emptyset$.*

Proof. By the chain rule of probability, it is sufficient to show that given $|A_k(V \setminus S)| \leq \ell_k/2$, with probability at least $1 - 1/(2N^{c+3})$ we have $A_k(S) \setminus A_k(V \setminus S) \neq \emptyset$. In this scenario, there are at least $\ell_k/2$ names that are not in $A_k(V \setminus S)$, and each node in S has a probability of at least $1/2$ of choosing such a name. After selecting a name, each node in S selects itself with probability p_k; in order for $A_k(S) \setminus A_k(V \setminus S)$ to be non-empty, at least one node in S must choose a name that is not in $A_k(V \setminus S)$ and then select itself.

The probability that a specific node $v \in S$ chooses a name not in $A_k(V \setminus S)$ and also selects itself is at least $p_k/2$. Since $|S| \geq \tau_k$, the expected number of nodes that generate a token outside $A_k(V \setminus S)$ is at least $(p_k/2)\tau_k \geq (\ell_k/2^{k+3})(\ell_k/\beta) = \ell_k^2/(2^{k+3}\beta)$, and by Chernoff,

$$\Pr[A_k(S) \setminus A_k(V \setminus S) = \emptyset] \leq e^{-\mathbb{E}[|A_k(S) \setminus A_k(V \setminus S)|]/2} \leq e^{-\ell_k^2/(2^{k+4}\beta)}$$
$$\leq e^{-(c+3)\ln N - 1} < \frac{1}{2N^{c+3}}.$$

□

Corollary 6.15. *Event* G2 *occurs with probability at least* $1 - 1/(2N^c)$.

Proof. For a specific node $u \in V$, level k and distance $d \geq \tau_k$, if $\Upsilon^d(u) \neq V$ then by strong connectivity we have $\Upsilon^d(u) \setminus \Upsilon^{d-\tau_k}(u) \geq \tau_k$. Let $S := \Upsilon^d(u) \setminus \Upsilon^{d-\tau_k}(u)$. If the condition in G2 fails to hold for u, k and d, then we have both $|A_k(V \setminus S)| \geq |A_k(\Upsilon^{d-\tau_k}(u))| \geq \ell_k/2$ and $A_k(S) \setminus A_k(V \setminus S) \subseteq A_k(\Upsilon^d(u)) \setminus A_k(\Upsilon^{d-\tau_k}(u)) = \emptyset$. By Lemma 6.14 this occurs with probability at most $1/(2N^{(c+3)})$. Since there are at most N nodes, at most $\log N < N$ levels and at most N distances to consider, a union bound shows that G2 is satisfied with probability at least $1 - N^3/(2N^{c+3}) = 1 - 1/(2N^c)$. □

Lemma 6.16. *With probability at least* $1 - 1/(2N^c)$ *we have* $|T_{\hat{k}}| \leq \ell_{\hat{k}}/2$.

Proof. By definition, $\hat{k} = \max\{\log\log N, \log n\}$, and therefore $2^{\hat{k}} \geq n$. Each node selects itself on level $\hat{k}$ with probability $p_{\hat{k}} = \ell_{\hat{k}}/2^{\hat{k}+2}$, so the expected number of nodes selected is $p_{\hat{k}} \cdot n \leq (\ell_{\hat{k}}/2^{\hat{k}+2}) \cdot 2^{\hat{k}} < \ell_{\hat{k}}/4$. By Chernoff, the probability that more than $\ell_{\hat{k}}/2$ nodes select themselves is at most $e^{-(1/3)\cdot\ell_{\hat{k}}/4} = e^{-\ell_{\hat{k}}/12} < 1/(2N^c)$. The number of tokens generated cannot exceed the number of nodes that selected themselves, so whenever no more than $\ell_{\hat{k}}/2$ nodes select themselves we also have $|T_{\hat{k}}| \leq \ell_{\hat{k}}/2$. The claim follows. □

Corollary 6.17. *Event* G *holds with probability at least* $1 - 1/N^c$.

Proof. This follows from Corollary 6.15 and Lemma 6.14 by union bound. □

Remark 6.18. In [96], where these results were first presented, we used a different selection scheme: instead of renaming and then having a subset of nodes select themselves, we had each node select a subset of tokens $\{1, \ldots, \ell_k\}$, with each token included in the set with probability roughly $1/2^k$. The ultimate effect is roughly the same as in the scheme presented above, as each node still contributes $\ell_k/2^k \approx \sqrt{2^k}$ tokens in expectation.

6.3.3 Solving HF_n using Token Selection

Next we put the token scheme we developed in 6.3.2 into use to solve the HF_n task.

As we explained in Section 6.3.1, our algorithm is composed of two components that run side-by-side. The first component is called the *estimation protocol*, and it is responsible for identifying a time t such that $\Upsilon^t(u) = V$; after time t, we say that node u is *ready to halt* (or simply *ready*). The second component is the *termination protocol*, responsible for ensuring that no node halts before all nodes are ready to halt. The interaction between the two components is limited to having the estimation protocol inform the termination protocol when the node is ready. Nevertheless, the success of each of the two protocols depends on the other.

In the estimation protocol, nodes initially generate tokens using the token selection scheme, which we refer to by a subroutine generate(k). The tokens are stored in pairs of the form (k, i), where k is the *level* on which the token was generated, and $i \in \{1, \ldots, \ell_k\}$ is the name of the token. The tokens are disseminated using pipelining, packing β tokens into each message, with lower-level tokens taking precedence over higher-level tokens. If a node does not receive any new tokens for $2\tau_k$ rounds, and if a total of no more than $\ell_k/2$ tokens were received on level k, then the node decides it has heard from all nodes and changes its status to *ready*. This condition is called the *level-k readiness condition*.

In the termination protocol, nodes compute the last time when some node was not ready, by sending each other *reset*(t) messages where t is the last time they have heard of where some node was not ready. Each node u maintains a variable M_u which stores the maximum value t received in a *reset*(t) message. If node u is *not* ready at time t, it sends a *reset*(t) message in round $t+1$; otherwise it sends *reset*(M_u). Each node u remembers the time R_u when it becomes ready; when u is ready and the time t satisfies $t - M_u > R_u$, node u halts.

The two sub-protocols interact with each other through the *ready* variable, which the estimation protocol sets to 1 when it completes; the R variable, which the estimation protocol sets to the current time when it completes; and the t variable, which stores the current time and is incremented by the termination protocol after each round.

We will now show that whenever event G occurs, the algorithm above successfully solves HF_n in $\tilde{O}(D + \sqrt{n/B})$ rounds.

Pipelining latency guarantee. The algorithm relies on pipelining to quickly disseminate small tokens throughout the network. Because we forward small tokens

Algorithm 6.1: A $(D + \tilde{O}(\sqrt{n/B}))$-round algorithm for HF_n: the estimation protocol

```
1  $ready \leftarrow 0$
2  for $k = \lceil \log\log N \rceil, \lceil \log\log N \rceil + 1, \ldots, \lceil \log N \rceil$ do
3      $Tokens \leftarrow \{k\} \times \text{generate}(k)$
4      $last\_update_k \leftarrow 0$
5  end
6  $Sent \leftarrow \emptyset$
7  for $r = 1, 2, \ldots$ do
8      $X \leftarrow$ select the $\beta$ smallest tokens in $Tokens \setminus Sent$
9      broadcast $X$ and set $Sent \leftarrow Sent \cup X$
10     receive tokens $Y$ from neighbors
11     for all $y = (k, i) \in Y \setminus Tokens$ do $\forall k' \geq k : last\_update_{k'} \leftarrow r$
12     $Tokens \leftarrow Tokens \cup Y$
13     if $\exists k : (|\{(k,i) \in Tokens\}| \leq \ell_k/2) \wedge (r - last\_update_k \geq 2\tau_k)$ then
       $ready \leftarrow 1$
14     $R \leftarrow t$
15     break
16 end
```

Algorithm 6.2: A $(D + \tilde{O}(\sqrt{n/B}))$-round algorithm for HF_n: the termination protocol

```
1  $M \leftarrow 0$
2  $R \leftarrow \infty$
3  $t \leftarrow 0$
4  while $ready = 0$ or $t - M \leq R$ do
5      if $ready = 0$ then
6          $M \leftarrow t$
7      end
8      send $reset(M)$
9      receive $reset(x_1), \ldots, reset(x_m)$ from neighbors
10     $M \leftarrow \max\{M, x_1, \ldots, x_m\}$
11     $t \leftarrow t + 1$
12 end
13 halt
```

before large ones, the progress of a token (k, i) can only be impeded by tokens on its own level (k) or lower levels ($k' < k$); there are at most $L_k = \sum_{i=1}^{k} \ell_i$ such tokens, and β of them can be sent per message. Thus the "latency" of token (k, i) is at most $\lceil L_k/\beta \rceil = \tau_k$.

Let $\mathit{Tokens}_v(t)$ stand for the value of the local variable *Tokens* at node v and time t. We show the following generalization of Lemma 3.18 (extending it to messages that each contain multiple tokens).

Lemma 6.19 (Pipelining Lemma II). *Assume that no node halts before time t. Then for any node v and token x, if $t \geq \mathrm{tdist}(x, v)$, either $x \in \mathit{Sent}_v(t)$ or $\mathit{Sent}_v(t)$ contains at least $\beta\,(t - \mathrm{tdist}(x, v))$ tokens that are smaller than x.*

Proof. The proof is by induction on t. The base case, $t = 0$, is immediate. For the induction step, assume that $t > 0$, and that the claim holds at time $t - 1$.

First we dispense with the case where $\mathrm{tdist}(x, v) = 0$, i.e., $x \in \mathit{Tokens}_v(0)$. Since node v initially has token x, if v has not sent x by time t, it must be that v was busy sending smaller tokens until time t; therefore $\mathit{Sent}_v(t)$ contains $\beta \cdot t = \beta(t - \mathrm{tdist}(x, v))$ smaller tokens.

Now suppose that $\mathrm{tdist}(x, v) > 0$, and let u be the next node from v along a shortest path from some node that initially knows x to v. From the induction hypothesis at $(v, t - 1)$,

($\star$) Either $x \in \mathit{Sent}_v(t - 1)$ or $\mathit{Sent}_v(t - 1)$ contains $\beta\,(t - 1 - \mathrm{tdist}(x, v))$ tokens smaller than x.

From the induction hypothesis at $(u, t - 1)$, together with the fact that $\mathit{Sent}_u(t - 1) \subseteq \mathit{Tokens}_v(t - 1)$ (because u is an in-neighbor of v),

($\star\star$) Either $x \in \mathit{Tokens}_v(t - 1)$ or $\mathit{Tokens}_v(t - 1)$ contains $\beta\,(t - 1 - \mathrm{tdist}(x, u)) = \beta\,(t - \mathrm{tdist}(x, v))$ tokens smaller than x.

Note that if $x \in \mathit{Sent}_v(t - 1)$, or if $\mathit{Sent}_v(t - 1)$ contains at least $\beta\,(t - \mathrm{tdist}(x, v))$ tokens smaller than t, then we are done, because $\mathit{Sent}_v(t - 1) \subseteq \mathit{Sent}_v(t)$. Therefore let us assume that $x \notin \mathit{Sent}_v(t - 1)$ and $\mathit{Sent}_v(t - 1)$ contains less than $\beta\,(t - \mathrm{tdist}(x, v))$ tokens smaller than x. There are three cases:

(a) $x \in \mathit{Tokens}_v(t - 1)$ and x is one of the β smallest tokens in $\mathit{Tokens}_v(t - 1) \setminus \mathit{Sent}_v(t - 1)$. In this case x is sent out in round t and added to $\mathit{Sent}_v(t)$, and the claim holds.

(b) $x \in \mathit{Tokens}_v(t - 1)$ but x is not one of the β smallest tokens in $\mathit{Tokens}_v(t - 1) \setminus \mathit{Sent}_v(t - 1)$. In this case β tokens smaller than x, which are not in $\mathit{Sent}_v(t - 1)$, are sent out in round t. It follows from this together with ($\star$) that $\mathit{Sent}_v(t)$ contains at least $\beta\,(t - \mathrm{tdist}(x, v))$ tokens smaller than x, so the claim holds.

(c) $x \notin \mathit{Tokens}_v(t - 1)$. Let m_T and m_S be the number of tokens smaller than x in $\mathit{Tokens}_v(t - 1)$ and $\mathit{Sent}_v(t - 1)$, respectively. Then ($\star\star$) indicates that $m_T \geq \beta\,(t - \mathrm{tdist}(x, v))$, and ($\star$) shows that $m_S \geq \beta\,(t - 1 - \mathrm{tdist}(x, v))$. In round t, node v sends at least $\min\{\beta, m_T - m_S\}$ tokens smaller than x which are not in $Sent_v(t-1)$. Therefore $Sent_v(t)$ contains at least $m_S + \min\{\beta, m_T - m_S\} = \min\{m_S + \beta, m_T\} \geq \beta\,(t - \text{tdist}(x, v))$ tokens smaller than x.

This completes the proof. □

Corollary 6.20. *For all $v \in V$ and $t \geq \tau_k$ we have*

$$Tokens_v(t) \subseteq A(\Upsilon^t(v)). \tag{6.3.1}$$

Moreover, if no node halts before time t, then

$$A_k(\Upsilon^{t-\tau_k}(v)) \subseteq Tokens_v(t) \subseteq A(\Upsilon^t(v)). \tag{6.3.2}$$

Proof. To show (6.3.2), let $x \in A_k(\Upsilon_v^{t-\tau_k})$, i.e., $\text{tdist}(x, v) \leq t - \tau_k$. Recall that $\tau_k = \lceil L_k/\beta \rceil$. By Lemma 6.19, either $x \in Sent_v(t)$ or $Sent_v(t)$ contains at least $\beta(t - \text{tdist}(x, v)) \geq \beta(t - (t - \lceil L_k/\beta \rceil)) \geq L_k$ tokens smaller than x. However, there are fewer than L_k tokens smaller than any token x on level at most k, so the second case is impossible. Therefore $x \in Sent_v(t) \subseteq Tokens_v(t)$.

The other inclusion, (6.3.1), is immediate: if $x \in Tokens_v(t)$ then we must have $\text{tdist}(x, v) \leq t$, otherwise it is impossible for node v to receive token x and add it to $Tokens_v$. □

Global invariant. Now we can show that when G occurs, the algorithm succeeds and has good round complexity. The following invariant tracks both the safety and the progress properties of the algorithm. When we write R_v without a time (that is, R_v rather than $R_v(t)$), we are referring to the time at which node v becomes ready.

Lemma 6.21. *If event G occurs, then for all $v \in V$ and times t,*

(a) If the level-k readiness condition first holds for v at time t (that is, if $R_v = t$), then $\Upsilon^t(v) = V$.

(b) If no node halts before time t, then

$$M_v(t) = \max\{t' \mid \exists w : (w, t') \in \mathsf{past}(v, t) \wedge ready_w(t') = 0\}.$$

(c) If no node halts before time t, and node v halts at time t, then for all $w \in V$ we have $(w, R_w) \in \mathsf{past}(v, t)$.

(d) If node v halts at time t then all nodes are ready at time t.

(e) If no node halts before time t, and node v halts at time t, then

$$M_v(t) = \max\{R_w \mid w \in V\} - 1.$$

Proof. The proof is by induction on t. The base case, $t = 0$, is immediate, because no node halts at time 0, the readiness condition does not hold for any node, and $M_v(0) = 0$ satisfies requirement (b).

Now suppose that the claim holds at all nodes until time $t - 1$ for all nodes, and let us show that the claim continues to hold for node v at time t.

(a) Suppose that the level-k readiness condition first holds for node v at time t. Then node v is not ready at time $t - 1$. By the induction hypothesis part (d), no node halts before time t. This allows us to apply Corollary 6.20.

The level-k readiness condition asserts that no new level-k tokens are received during the time interval $[t - 2\tau_k, t]$. Assume for the sake of contradiction that $\Upsilon^t(v) \neq V$, and let $S := \Upsilon^{t-2\tau_k}(v), S' := \Upsilon^{t-\tau_k}(v)$. From Corollary 6.20 we see that

(I) $A_k(S') \subseteq \mathit{Tokens}_v(t)$, that is, all tokens generated by the nodes of S' are known to v at time t; and

(II) $\mathit{Tokens}_v(t - 2\tau_k) \subseteq A_k(S)$, i.e., at time $t - 2\tau_k$ node v only knows tokens generated by the nodes of S.

Since no new tokens were added to Tokens_v between time $t - 2\tau_k$ and time t, we must have $A_k(S') = A_k(S)$; in other words, the nodes of $S' \setminus S$ did not generate any tokens that were not already generated by the nodes of S. Also, there can be at most $\ell_k/2$ tokens in $A_k(S)$, because $A_k(S) \subseteq A_k(S') \subseteq \mathit{Tokens}_v(t)$ and the second part of the readiness criterion asserts that $|\mathit{Tokens}_v(t)| \leq \ell_k/2$. But this violates $\mathsf{G2}$ (with parameter $d = t - \tau_k$), and thus contradicts our assumption that event G occurs.

(b) Suppose that no node halts before time t. Then in particular, no node halts before time $t - 1$, and we can apply part (b) of the induction hypothesis. According to the termination protocol, in round t, each node $w \in \Upsilon(v)$ sends $\mathit{reset}(M_w(t-1))$; by part (b) of the induction hypothesis, for each $w \in \Upsilon(v)$,

$$M_w(t-1) = \max\{t' \mid \exists w : (w, t') \in \mathsf{past}(w, t-1) \wedge \mathit{ready}_w(t') = 0\}.$$

It follows that

$$\begin{aligned} M_v(t) &= \max\{M_w(t-1) \mid w \in \Upsilon(v)\} \\ &= \max\{t' \mid \exists w, w' : w \in \Upsilon(v) \wedge (w', t') \in \mathsf{past}(w', t-1) \wedge \mathit{ready}_{w'}(t') = 0\} \\ &= \max\{t' \mid \exists w' : (w', t') \in \mathsf{past}(v, t) \wedge \mathit{ready}_{w'}(t') = 0\}. \end{aligned}$$

(c) Suppose that node v halts at time t. Then:

(I) $\mathit{ready}_v(t) = 1$, that is, $t \geq R_v$. By part (a) of the induction hypothesis,

$$\Upsilon^{R_v}(v) = V, \tag{6.3.3}$$

that is, the inwards-eccentricity of node v does not exceed R_v.

(II) Moreover,

$$t - M_v(t) > R_v. \tag{6.3.4}$$

From (6.3.3) together with the fact that no node halts before time t and $t \geq R_v$ we have $\mathsf{past}(v,t)_{t-R_v} = V$, that is, for each $w \in V$, $(w, t - R_v) \in \mathsf{past}(v,t)$. For each $w \in V$, let $t_w \geq t - R_v$ be the maximal time such that $(w, t_w) \in \mathsf{past}(v,t)$. If $t_w < R_w$, that is $ready_w(t_w) = 0$, then by part (b) we have

$$M_v(t) = \max \{t' \mid \exists w' : (w', t') \in \mathsf{past}(v,t) \wedge ready_{w'}(t') = 0\} \geq t_w \geq t - R_v. \tag{6.3.5}$$

But this contradicts (6.3.4), which asserts that $M_v(t) < t - R_v$. It follows that $t_w \geq R_w$ for all $w \in V$, and therefore we have $(w, R_w) \rightsquigarrow (w, t_w) \rightsquigarrow (v,t)$ and $(w, R_w) \in \mathsf{past}(v,t)$.

(d) Suppose that node v halts at time t. If node v is not the first node to halt — that is, if there is some node that halts at some time $t' < t$ — then by part (d) of the induction hypothesis, all nodes are ready at time t', and they remain ready at time t. Thus, assume that no node halts before time t. By part (c), which we have already proven for time t, we have $(w, R_w) \in \mathsf{past}(v,t)$ for each $w \in V$. But this implies that $t \geq R_w$ for all $w \in V$, that is, all nodes are ready by time t.

(e) Suppose that v halts at time t, and no node halts before time t. Let $t' < t$ be the last time when some node w is not ready. Then $\max\{R_{w'} \mid w' \in V\} = R_w = t'+1$. By part (c), $(w, R_w) \in \mathsf{past}(v,t)$, and therefore $(w, t') \in \mathsf{past}(v,t)$ as well (because $t' = R_w - 1$). By part (b),

$$M_v(t) = \max \{t' \mid \exists w' : (w', t') \in \mathsf{past}(v,t) \wedge ready_{w'}(t') = 0\},$$

Since $(w, t') \in \mathsf{past}(v,t)$ and $ready_w(t') = 0$, it follows that $M_v(t) \geq t'$. On the other hand, there does not exist any node $w' \in V$ such that $ready_{w'}(s) = 1$ for some $s > t'$. Therefore $M_V(t) \leq t'$ as well, and together we have $M_v(t) = t' = \max\{R_{w'} \mid w' \in V\} - 1$.

□

Lemma 6.22. *For all nodes v and times t we have*

$$M_v(t) < \max\{R_w(t) \mid w \in V\}.$$

Proof. First, note that if some node w is not ready at time t, then the claim holds trivially, because $R_w(t) = \infty$. Thus, assume that all nodes are ready at time t, and let $t' < t$ be the last time when some node was not ready. Then $\max\{R_w(t) \mid w \in V\} = t' + 1$, that is, the last node becomes ready at time $t' + 1$.

By Lemma 6.21 part (c), no node halts before time t', and by part (b) of the lemma, for each $v \in V$ we have

$$M_v(t') = \max\{t'' \mid \exists w : (w, t'') \in \mathsf{past}(v, t') \wedge ready_w(t'') = 0\} \leq t',$$

and therefore $\max\{M_v(t') \mid v \in V\} \leq t'$.

From time $t'+1$ onwards, all nodes are ready; they do not execute line 6 of the termination protocol, and do not increase their M variable except upon receiving a *reset*(x) value with $x > M$ from some neighbor. Therefore, an easy induction on time shows that for all $t \geq t'+1$ and nodes v we have $M_v(t) \leq \max\{M_w(t') \mid w \in V\} \leq t'$. The claim follows, because we already observed that $\max\{R_w(t) \mid w \in V\} = t'+1 > t'$. □

Whenever event G occurs, we can also bound the round complexity of the algorithm in terms of $\hat{k}$, the "correct" value of k. We first show that all nodes become ready quickly, and then argue that they halt not too long after the last node becomes ready.

Lemma 6.23. *In graphs of diameter D, when event G occurs, then all nodes become ready by time $D + 3\lceil L_{\hat{k}}/\beta\rceil$. That is, for each $v \in V$ we have*

$$R_v \leq D + 3\lceil L_{\hat{k}}/\beta\rceil.$$

Proof. Suppose that there is some node that is not ready before time $D + 3\lceil L_{\hat{k}}/\beta\rceil$ (otherwise we are done). Then by Lemma 6.21, no node halts before this time, and we can apply Corollary 6.20 to all times $t' \leq D + 3\lceil L_{\hat{K}}/\beta\rceil$.

From Corollary 6.20, at time $D + \lceil L_{\hat{k}}/\beta\rceil$ we have $A_{\hat{k}}(\Upsilon_v^D) = A_{\hat{k}}(V) \subseteq \mathit{Tokens}_v(t)$, i.e., node v knows all $L_{\hat{k}}$ tokens of the form (k, i) where $k \leq \hat{k}$. After this time node v cannot hear any new tokens on level $\hat{k}$, so by time $D + 3L_{\hat{k}}/\beta$ the first part of the readiness criterion is satisfied for level $\hat{k}$. □

Lemma 6.24. *If the last node becomes ready at time t, then each node v halts no later than time $t + R_v$.*

Proof. Fix a node v. Since the last node becomes ready at time t, we have

$$\max\{R_w(t) \mid w \in V\} = t.$$

By Lemma 6.22,

$$M_v(t + R_v) < \max\{R_w(t) \mid w \in V\} = t.$$

Therefore $t + R_v - M_v(t + R_v) > R_v$. At time $t + R_v \geq R_v$ we have $\mathit{ready}_v = 1$, so the condition for terminating at time $t + R_v$ (line 4) holds at node v. □

Corollary 6.25. *When event G occurs, the algorithm terminates in $2D + 6\lceil L_{\hat{k}}/\beta\rceil$ rounds.*

Proof. By Lemma 6.23, all nodes v have $R_v \leq D + 3\lceil L_{\hat{k}}/\beta\rceil$. Applying Lemma 6.23 we see that each node v terminates no later than time $\max\{R_w \mid w \in V\} + R_v \leq 2(D + 3\lceil L_{\hat{k}}/\beta\rceil)$. □

Putting all the ingredients together, we obtain the following theorem.

Theorem 6.26. *For any constant c, with probability at least $1 - 1/N^c$ each node v halts at a time $t = O(D + L_{\hat{k}}/\beta) = \tilde{O}(D + \sqrt{n/B})$ such that $\Upsilon^t(v) = V$.*

Proof. By Corollary 6.17, event G occurs with probability at least $1 - 1/N^c$. As we have already seen, whenever $\mathsf{G} = \mathsf{G1} \wedge \mathsf{G2}$ occurs, the algorithm succeeds and halts after at most $2D + 6\lceil L_{\hat{k}}/\beta \rceil$ rounds. By definition, since $\hat{k} = \max\{\log\log N, \log n\}$,

$$L_{\hat{k}} = \sum_{i=1}^{\hat{k}} \ell_i = O(\sqrt{2^{\hat{k}}\beta \ln N}) = O(\sqrt{(n + \log N)\beta \log N}).$$

(The last term in the sum, $\ell_{\hat{k}}$, dominates.) Substituting into the running time yields

$$2D + O\left(\frac{\sqrt{n\beta}\log N}{\sqrt{\beta}}\right) = 2D + O\left(\sqrt{\frac{n}{\beta}}\log N\right).$$

We assumed that β tokens can fit in each B-bit message. Each token is a pair (k, i), where $k \leq \max\{\log\log N, \log n\} \leq \log N$ and $i \leq \ell_k = O(B \log N)$. Therefore each token can be represented using $O(\log\log N + \log B)$ bits, and $\beta = \Omega(B/(\log B + \log\log N))$. The running time is therefore

$$2D + O\left(\sqrt{\frac{n(\log B + \log\log N)}{B}}\log N\right) = 2D + \tilde{O}\left(\sqrt{\frac{n}{B}}\right).$$

□

6.3.4 Simultaneous HF_n

Algorithm 6.1 guarantees that each node u will halt at some time t_u such that $\Upsilon^{t_u}(u) = V$, but it does not guarantee that all nodes will halt at the same time. This has two drawbacks that make Algorithm 6.1 less useful as a building block: first, if nodes do not finish Algorithm 6.1 at the same time, then they do not begin executing whatever code follows it synchronously; and second, it may be useful for all nodes to obtain the same upper bound on the diameter of the graph, but the time t_u when Algorithm 6.1 halts is only an upper bound on node u's own inward-eccentricity.

It is easy to modify the algorithm to obtain simultaneous termination and also provide all nodes with the same upper bound on the diameter. To do this, we leave the estimation protocol as-is, and modify only the termination protocol. Now, when the termination condition (the negation of the condition in line 4) holds, nodes do not immediately halt; instead they increment M and start an M-round *countdown*, where for M rounds they inform all nodes to halt and adopt M as their final estimate. During the countdown all nodes that are already informed of the countdown send *countdown*(T, M) messages, where T is the target time for halting and M is the final estimate for the diameter. At time T, all nodes halt and output M.

Algorithm 6.3: Simultaneous HF_n: the termination protocol

```
1  $M \leftarrow 0$
2  $R \leftarrow \infty$
3  $t \leftarrow 0$
4  $H \leftarrow \infty$
5  while $ready = 0$ or $t - M \leq R$ do
6      if $ready = 0$ then
7          $M \leftarrow t$
8      end
9      send $reset(M)$
10     receive msgs from neighbors
11     if $countdown(t', x) \in$ msgs then
12         $H \leftarrow t'$
13         $M \leftarrow x$
14         goto 24
15     end
16     else if msgs $= \{reset(x_1), \ldots, reset(x_m)\}$ then
17         $M \leftarrow \max\{M, x_1, \ldots, x_m\}$
18     end
19     $t \leftarrow t + 1$
20 end
21 $M \leftarrow M + 1$
22 $H \leftarrow t + M$
23 while $t < H$ do
24     send $countdown(H, M)$
25     receive msgs from neighbors
26     if $countdown(H', M') \in$ msgs then
27         $H \leftarrow \min\{H, H'\}$
28     end
29     $t \leftarrow t + 1$
30 end
31 halt and output $M$
```

Analysis. As with Algorithm 6.1, we analyze the algorithm under the assumption that the "good" event G occurs. We already know from the previous section that G occurs with high probability.

We say that *node v starts the countdown at time t* if no *countdown* messages are sent until time t, and the first *countdown* message is sent by node u in round $t+1$.[2] As the countdown starts exactly at the time when in Algorithm 6.1 the first node decides to halt, Lemma 6.21 continues to hold, with the following modifications:

Lemma 6.27. *If event G occurs, then for all $v \in V$ and times t,*

(a) If the level-k readiness condition first holds for v at time t (that is, if $R_v = t$), then $\Upsilon^t(v) = V$.

(b) If no node halts before time t, then

$$M_v(t) = \max\{t' \mid \exists w : (w, t') \in \mathsf{past}(v, t) \wedge ready_w(t') = 0\}.$$

(c) If no node halts before time t, and node v ***starts the countdown*** *at time t, then for all $w \in V$ we have $(w, R_w) \in \mathsf{past}(v, t)$.*

(d) If node v ***starts the countdown*** *at time t then all nodes are ready at time t.*

(e) If no node halts before time t, and node v ***reaches line 20*** *at time t, then*

$$M_v(t) = \max\{R_w \mid w \in V\} - 1.$$

The proof is nearly identical to the proof of the original Lemma 6.21, and we do not repeat it here.

Corollary 6.28. *If node v is past line 20 of the algorithm at time t, then $M_v(t) = \max\{R_w \mid w \in V\}$.*

Proof. There are two ways a node can reach a line greater than 20:

I. It can execute line 20 and proceed from there. By Lemma 6.27, when node v reaches line 20 in round t, we have $M_v(t-1) = \max\{R_w \mid w \in V\} - 1$. Upon executing line 20 we obtain $M_v(t) = \max\{R_w \mid w \in V\}$.

II. Node v may receive a *countdown*(x, y) message in some preceding round $r \le t$ and set M_v to y as a result. In this case we have $y = M_u(r-1)$, where u is the node that sent the *countdown*(x, y) message. Node u is itself already past line 20 when it sends the *countdown*(x, y) message. Therefore an easy induction on time shows that the claim holds in this case as well.

□

[2] If more than one node sends a *countdown* message in round $t+1$, then all nodes that send such messages are said to start the countdown at time t.

Now we can show that the following invariant holds during the countdown:

Lemma 6.29. *Assume that event* G *occurs, and let* $v \in V$ *be a node that starts the countdown at time* t. *Then* $M_v(t) \geq D$, *and for any* $t' \in \{t, \ldots, t + M_v(t)\}$,

(a) No node halts before time t',

(b) All nodes $u \in V$ *have* $H_u(t') \geq H_v(t)$, *and*

(c) All nodes u *such that* $\mathrm{dist}(v, u) \leq t' - t$ *are past line 20 and have* $M_u(t') = M_v(t)$ *and* $H_u(t') = H_v(t) = t + M_v(t)$.

Proof. Given event G we have $\Upsilon^{R_w}(w) = V$ for all nodes $w \in V$, and consequently $\max\{R_w \mid w \in V\} \geq D$. If v starts the countdown at time t, then in particular no node halts until time t, and by Corollary 6.28 we have $M_v(t) = \max\{R_w \mid w \in V\} \geq D$.

Now let us show the invariant concerning times $t' \geq t$, by induction on t'. The base case is $t' = t$. At this time no node has halted yet, by our assumption that the countdown starts at time t, so part (a) holds. Also, since no *countdown* messages have been sent yet, any node u that has changed H_u from its initial value of ∞ is already past line 20; by Corollary 6.28, $M_u(t) = M_v(t) = \max\{R_w \mid w \in V\}$, and consequently $H_u(t) = t + M_u(t) = t + M_v(t) = H_v(t)$. Therefore part (b) holds as well. And finally, node v is the only node at distance 0 from itself, so part (c) holds trivially.

For the induction step, assume that the claim holds until time t', and consider time $t' + 1 \leq t + M_v(t)$. We need to show the following.

(a) No node has halted before time $t' + 1$: from part (b) of the induction hypothesis, all nodes u have $H_u(t') \geq H_v(t) = t + M_v(t) \geq t' + 1$. Nodes only halt at time s if $H_u(s) = s$, so no node can halt before time $t' + 1$.

(b) All nodes $u \in V$ have $H_u(t'+1) \geq H_v(t)$: by part (b) of the induction hypothesis, for any $u \in V$, $H_u(t') \geq H_v(t)$. If node u does not change the value of H_u in round $t' + 1$ then the claim continues to hold. If node u *does* change H_u to some value x in round $t' + 1$, there can be two reasons for the change:

   I. Node u received a *countdown*(x) message in round $t' + 1$ from some other node w: then $x = H_w(t')$, and by the induction hypothesis, $H_w(t') \geq H_v(t)$.

   II. Node u executes line 22 in round $t' + 1$, and sets H_u to $t' + M_u(t')$. As in the induction base, node u must have previously executed line 20, and by Lemma 6.27 part (e) we have $M_u(t') = M_v(t)$. Therefore $H_u(t' + 1) = t' + M_u(t') \geq t + M_v(t) = H_v(t)$.

(c) All nodes u such that $\mathrm{dist}(v, u) \leq t' + 1 - t$ have $M_u(t' + 1) = M_v(t)$ and $H_u(t'+1) = H_v(t) = t + M_v(t)$: fix one such node u. By the induction hypothesis we have $H_w(t') \geq H_v(t)$ for all nodes w, and by Corollary 6.28, any node w

that sends a *countdown* in round $t' + 1$ has $M_w(t') = M_v(t)$. Consequently all *countdown*(x, y) messages sent in the network have $x \geq H_v(t)$ and $y = M_v(t)$.

If we already have $M_u(t') = M_v(t)$ and $H_u(t') = H_v(t)$ (in particular, by the induction hypothesis, if $\text{dist}(v, u) \leq t' - t$), then no *countdown* message received in round $t' + 1$ can cause u to change either variable, as M is never changed after line 20, and H can only be changed upon receiving a smaller value in a *countdown* message (but we showed that no smaller values than $H_v(t)$ are sent).

If $\text{dist}(v, u) = t' - t + 1$, then there is some node $w \in \Upsilon(u)$ such that $\text{dist}(v, w) = t' - t$, and by the induction hypothesis, $M_w(t') = M_v(t)$ and $H_w(t') = H_v(t)$. In round $t' + 1$ node w sends a *countdown*$(H_w(t'), M_w(t'))$ message, which is received by node u. Upon receiving the message node u sets $M_u(t' + 1) \leftarrow H_w(t') = H_v(t)$ and, if it did not already have $M_u(t') = M_v(t)$, it now sets $M_u(t' + 1)$ to $M_w(t') = M_v(t)$.

□

Theorem 6.30. *For any constant c, with probability at least $1 - 1/N^c$, all nodes halt at the same time $t = O(D + L_k/\beta) = \tilde{O}(D + \sqrt{n/B})$ and output the same value M satisfying $M \geq D$.*

Proof. We already saw in Theorem 6.26 that with probability at least $1 - 1/N^c$, event G occurs, and in the original algorithm all nodes u halt by time $\tilde{O}(D + \sqrt{n/B})$ and have $R_u = \tilde{O}(D + \sqrt{n/B})$ as well. In the new algorithm the countdown starts at the time the first node would halt in the original algorithm, so at some time $t = \tilde{O}(D + \sqrt{n/B})$, some node v starts the countdown, with $M_v(t) = \max\{R_w \mid w \in V\} = \tilde{O}(D + \sqrt{n/B})$ and $H_v(t) = t + M_v(t) = \tilde{O}(D + \sqrt{n/B})$. By Lemma 6.29, all nodes halt at time $H_v(t)$ and output $M_v(t)$, which satisfies $M_v(t) \geq D$ by Corollary 6.28. □

Chapter 7

Distributed Task Allocation in Directed Networks

In many distributed systems, a large amount of work is performed quickly and efficiently by partitioning the work across the participants in the network. The "work" to be performed can range from computational work, such as simulating a complex physical environment or solving a complex optimization problem, to physical work, such as having robots travel to various locations to carry out assorted tasks. In all cases, effectively parceling out parts of the global goal to be performed by individual participants is key to the overall efficiency of the system. This problem has been studied in many forms and guises, from fault-tolerant task allocation (see [137]) to centralized and distributed scheduling (e.g., [57, 38, 109] and many others), from theoretical (e.g., the k-server problem [52]) to practical approaches [1].

In this chapter we study the *communication complexity* of task allocation, that is, the total number of bits that the participants need to exchange to allocate the tasks between themselves. We consider an abstract version of the problem, $\textsc{TaskAlloc}_{m,n}$, where m players must jointly perform a set of n tasks (we often assume that the set of tasks is $\{1, \dots, n\}$). Each player i receives as input a set $X_i \subseteq \{1, \dots, n\}$ of tasks that it is capable of performing; for example, in the case of robots performing tasks at different geographical locations, the player's input might consist of the set of locations requiring servicing that the robot can "see" with its sensors. The goal is for the players to partition the tasks between them: each player i must output a subset of tasks $Y_i \subseteq X_i$, such that $\bigcup_i Y_i = \{1, \dots, n\}$ and $Y_i \cap Y_j = \emptyset$ for all $i \neq j$. To make the problem feasible, we consider only inputs $X_1, \dots, X_m$ such that $\bigcup_i X_i = \{1, \dots, n\}$, that is, there *exists* some partition that covers all the tasks. The players are charged for communicating among themselves, but not for writing their output sets $Y_1, \dots, Y_m$. We consider various models of communication between the players, from shared-blackboard to arbitrary strongly-connected communication networks.

The task allocation problem can be viewed as a restricted one-shot instance of the well-known k-server problem [52], where a centralized online algorithm assigns tasks to k servers, minimizing the total cost of servicing all tasks. In the k-server problem each (server, task) pair is associated with a *cost* for having the server perform the task, and moreover, tasks arrive continually and must be assigned in an online manner. In TaskAlloc, all tasks are initially known, and all have a cost of either 1 (if the task can be performed by the player) or ∞ (if it cannot). Partitioning the tasks between the players corresponds to finding a minimum-weight assignment of tasks to servers. To the best of our knowledge, the k-server problem has not been studied from the perspective of communication complexity, although distributed variants have been studied (e.g., [13, 20]). These variants achieve good competitive ratio w.r.t. the cost and the number of messages sent, but require large messages and total communication. (The algorithm of [20] is competitive w.r.t. the number of messages sent, but not the size of individual messages; these can grow large, as serves simulate a centralized k-server algorithm and send each other configurations of the centralized algorithm.) In this thesis we are not interested in competitive analysis, as the variant we consider is single-shot; extensions to weighted inputs and the online setting remain interesting directions for future work.

Our work raises several open problems, which we discuss in Section 7.7. In general, two-player communication complexity lower bounds have proven very useful in proving lower bounds on various distributed problems (e.g., [123, 133, 58, 96]). However, distributed computation is more accurately captured by *multi-player* communication games in the number-in-hand model, where each player knows its own input (contrast with the number-on-forehead model, where each player knows all the *other* players' input). The number-in-hand model was neglected by the communication complexity community for a time, but recently several new techniques have led to exciting advances (see [65, 77, 78]). We believe that the complexity of distributed computing in the CONGEST model, where bandwidth is restricted, can be analyzed in terms of a multi-player communication game. Importing problems from the distributed computing world into the communication complexity model raises issues which are not often considered in existing communication complexity lower bounds: *search problems*, where players are allowed to choose one of many possible outputs (e.g., electing a leader or reaching consensus);[1] *partial knowledge*, where each player needs to output only part of the answer (as exemplified in the TaskAlloc problem); and *unicast communication cost*, where we wish to charge players for the number of other players they communicate with, not just the total communication complexity as in the shared blackboard model. We believe that these issues yield new and interesting questions in multi-player communication complexity.

Despite the apparent simplicity of the problem, TaskAlloc is rich enough to admit a strong lower bound: in Section 7.2 we show that even for two players (using public random coins), $\text{TaskAlloc}_{2,n}$ has a randomized communication complexity of $\Omega(n)$. We apply this bound in Section 7.3 to show that computing a rooted spanning tree in directed broadcast networks with diameter 2, where each message is restricted to B bits, requires $\Omega(n/B)$ rounds — even when the size of the network is fixed in advance, the diameter is known, and nodes have unique identifiers in the range

[1]Many well-known lower bounds in communication complexity concern decision problems, but there are some cases where search problems play an important role; for example, in [81] it is shown that proving a circuit-depth lower bound for a function f reduces to proving a communication lower bound on a certain search problem associated with f.

$1, \ldots, n$. In Section 7.4 we study the communication complexity of $\textsc{TaskAlloc}_{m,n}$ in the classical multi-player setting where players communicate over a shared blackboard. We give two randomized algorithms: the first requires players to write large messages on the blackboard, but has an overall communication complexity of $O(n \log(n+m))$ and terminates in $O(\log m)$ rounds with high probability on any input. The second algorithm we give works well when no small number of players has to take care of a large set of tasks, a property that is formally captured by our definition of *task-player expansion* in Section 7.1. For inputs with task-player expansion α, our second randomized algorithm terminates in $O\big((1/\alpha + \log m) \log n\big)$ rounds (or better, see Theorem 7.13 in Section 7.4) and uses messages of size $O(\log(n+m))$. This is optimal to within a polylog(m,n) factor. In Section 7.5 we extend our results to an arbitrary strongly-connected communication network between the players, with the size of individual messages bounded by B. We show that in networks of diameter D, $\textsc{TaskAlloc}_{m,n}$ can be solved in $O(D + \sqrt{m/B} \log m + n \log(n+m)/B)$ rounds, using a total of $O((m+n)\,\text{polylog}(m,n))$ bits of communication.

7.1 Problem Statement

Task allocation. The *distributed task allocation* problem, denoted $\textsc{TaskAlloc}_{m,n}$, is defined over a set T of $|T| = n$ tasks and a set V of $|V| = m$ players. We often assume that $T = \{1, \ldots, n\}$. Each player $v \in V$ receives an input set $X_v \subseteq T$, with the promise that $\bigcup_{v \in V} X_v = T$. The goal is for each player to output a set $Y_v \subseteq X_v$, such that for all $u, v \in V$ we have $Y_u \cap Y_v = \emptyset$, and moreover, $\bigcup_{v \in V} Y_v = T$ (that is, $\{Y_v \mid v \in V\}$ is a partition of T).

The input assignment $\{X_v \mid v \in V\}$ induces a bipartite graph $H = (V \dot{\cup} T, F)$, where the edges F are given by $F := \{(v, x) \in V \times T \mid x \in X_v\}$. We call H the *task-player graph*. For convenience, for each task $i \in T$, we let $V_i := \{v \in V \mid i \in X_v\}$ denote the set of players that have task i in their input.

Communication model. In order to solve a given task allocation instance the players in V must communicate. In the current paper we assume synchronous communication, i.e., the players proceed in synchronous rounds. We consider two models of communication:

- In the classical *shared blackboard* model, the players communicate by writing messages on a shared blackboard which is visible to all other players.
- In addition, we are interested in the *directed network model*, which is the same model we studied in Chapter 6. We assume that initially the players do not know anything about the graph except possibly its size, i.e., the number of players.

The shared blackboard model is a special case of the general network model, obtained by choosing $G := K_m$ (the complete graph on m nodes).

We are interested in the following performance measures:

- *Total communication complexity*: the total number of bits ever sent or written on the blackboard during the protocol.

- *Round complexity*: how many rounds of communication are required (where in each round each player can send/write one message).
- *Message size*: how many bits the players send or write on the blackboard in each round. This parameter is usually specified as an external constraint, and we denote it by B.

Task-player expansion. The hardness of an instance of TASKALLOC depends on the properties of the input assignment, represented by the task-player graph H. In particular, we will show that the complexity depends on how well tasks can be distributed among the players, as formally captured by the following definition.

Definition 7.1 (Task-Player Expansion). *The* task-player expansion *of a task-player graph $H = (V \dot{\cup} T, F)$ is defined as*

$$\alpha(H) := \min_{T' \subseteq T} \frac{\left|\bigcup_{x \in T'} V_x\right|}{|T'|}.$$

Informally, when the task-player expansion is large, each set of tasks can be assigned to many different players; the problem is in some sense less constrained, which makes it easier to solve. The smallest value $\alpha(H)$ can take is $1/n$, which occurs when one player receives all the tasks in his input and the others receive nothing. The largest value, obtained when H is the complete bipartite graph, is m/n.

7.2 Two-Player Lower Bound for Task Allocation

We begin by analyzing the complexity of task allocation in the classic two-party model, where two players, Alice and Bob, wish to allocate n tasks between them. In this section we let TASKALLOC$_n$ stand for TASKALLOC$_{2,n}$, and we use U, V to denote the inputs to Alice and Bob respectively and A, B to denote Alice and Bob's outputs. It is assumed that $T = [n]$ and both players know n.

The tasks over which Alice and Bob "contend" are the ones in the intersection of their inputs, $U \cap V$; these tasks must be output by one player but cannot be output by both. If Alice does *not* output some task that she received in her input, then she must know that this task is in the intersection of the inputs, and that Bob will output it. The connection between task allocation and finding the intersection of the players' inputs is formalized in the following easy lemma:

Lemma 7.2. *Let (A, B) be a valid output on instance (U, V) of* TASKALLOC$_n$*; that is, $A \cup B = [n]$, $A \cap B = \emptyset$, $A \subseteq U$, and $B \subseteq V$. Then*

(a) $U \setminus A \subseteq U \cap V$ and $V \setminus B \subseteq U \cap V$; and

(b) $U \cap V \subseteq (U \setminus A) \cup (V \setminus B)$.

Proof. For (a), let $x \in U \setminus A$ (the other inclusion is similar). In particular, then, $x \notin A$, and since $A \cup B = [n]$, we must have $x \in B$. But $B \subseteq V$, and therefore $x \in U \cap V$.

For (b), let $x \in U \cap V$. Since $A \cup B = [n]$ we have $x \in A \cup B$; assume w.l.o.g. that $x \in A$. Because $A \cap B = \emptyset$ we have $x \notin B$, but on the other hand we have $x \in V$ (as $x \in U \cap V$). Together we have $x \in V \setminus B$. □

Theorem 7.3. *The randomized public-coin communication complexity of two-player task allocation is* $\Omega(n)$.

Proof. Suppose the input (U, V) is generated according to the following distribution $\mathcal{D}$. Choose a random subset $X \subseteq [n]$ of size pn. Next, choose a random subset $Y \subseteq [n] \setminus X$ of size $(1-p)n/2$. Let $U := X \cup Y$ and let $V := X \cup \bar{Y}$. Let $\mathcal{C}$ be the support of the distribution (i.e., pairs where each set is of size $(1+p)n/2$, and the intersection is of size pn).

We are interested in the probability over $(U, V) \sim \mathcal{D}$ that given only U, Alice can guess $U \cap V = X$. Given U, the set X is a uniformly-chosen subset of pn elements of U. Therefore, given U, Alice's chance of guessing X is at best

$$\binom{(1+p)n/2}{pn}^{-1} \leq \left(\frac{(1+p)n/2}{pn}\right)^{-pn} = \left(\frac{1+p}{2p}\right)^{-pn}. \tag{7.2.1}$$

Informally, we will show that if there exists a protocol P for task allocation with communication complexity $o(n)$, then Alice can guess $X = U \cap V$ with statistically impossible accuracy (i.e., she can succeed with probability better than the bound above). For an execution of P, let A and B be the outputs of Alice and Bob, respectively. By Lemma 7.2, $U \setminus A \subseteq U \cap V$; in other words, after executing P, Alice knows that each element in her input that she did not output is in $U \cap V$. If $U \setminus A$ is large, this provides Alice with enough information to guess the remaining elements of $U \cap V$ "too accurately".

More formally, let P be a public-coin protocol for task allocation with t rounds, which succeeds with probability at least $1/2$ on each input. For each input $(U, V) \in \mathcal{C}$ we have $|U \cap V| = pn$ (by definition of $\mathcal{C}$), and by Lemma 7.2, $|(U \setminus A) \cup (V \setminus B)| \geq pn$ (in fact the lemma shows equality, but we do not require it here). Thus, if P succeeds then either $|U \setminus A| \geq pn/2$ or $|V \setminus B| \geq pn/2$; assume w.l.o.g. that with probability at least $1/4$ over both the choice of (U, V) and the coin tosses of P, the players eventually output a correct output (A, B) with $|U \setminus A| \geq pn/2$.

Now Alice can guess $X = U \cap V$ given U as follows: she simulates protocol P by guessing a t-bit transcript (in addition to P's own randomness), obtaining some output A. With probability at least $1/4 \cdot 2^{-t}$, Alice guesses a transcript for P that

(a) Matches the input and P's randomness (that is, a transcript that might actually be generated when P is executed with input (U, V) and the specific random string Alice guessed), and

(b) Leads to a correct output (A, B), so in particular $U \setminus A \subseteq X$, and

(c) Leads to an output (A, B) such that $|U \setminus A| \geq pn/2$.

Now Alice knows that each element of $U \setminus A$ is in the intersection X (by Lemma 7.2), and there are at least $pn/2$ such elements. This considerably narrows down the number of possibilities for X: there remain at most

$$\binom{(1+p)n/2 - |U \setminus A|}{pn - |U \setminus A|} \leq \binom{(1+p)n/2}{pn/2}$$
$$\leq \left(\frac{(1+p)en/2}{pn/2}\right)^{pn/2} = \left(\frac{(1+p)e}{p}\right)^{pn/2}$$

values for X that are consistent with $U \setminus A \subseteq X$. By choosing the most likely possibility given the transcript and the public randomness, Alice can guess the correct value of X with probability at least

$$\left(\frac{(1+p)e}{p}\right)^{-pn/2} = \left(\frac{1+p}{2p} \cdot 2e\right)^{-pn/2}.$$

Combining with the upper bound from (7.2.1) on Alice's success probability, we obtain

$$\frac{1}{4} \cdot 2^{-t} \cdot \left(\frac{1+p}{2p} \cdot 2e\right)^{-pn/2} \leq \left(\frac{1+p}{2p}\right)^{-pn}.$$

Simplifying yields

$$t \geq \frac{pn}{2}\left(\log\frac{1+p}{2p} - (1 + \log e)\right) - 2.$$

To obtain a non-trivial lower bound we must select p such that $\log\frac{1+p}{2p} > (1 + \log e) \approx 2.4$. For example, $p := 1/16$ satisfies this constraint.

To conclude, if P is a protocol for 2-player task allocation, then there must exist an input on which with probability at least $1/2$, at least $\frac{n}{32}\left(\log\frac{17}{2} - (1+\log e)\right) - 2 = \Omega(n)$ bits are exchanged. Therefore the worst-case expected communication complexity of P is $\Omega(n)$. □

Remark 1. The lower bound can be extended to a relaxed variant of TaskAlloc, where we allow an ε-fraction of tasks to be assigned to *both* players for a sufficiently small constant $\varepsilon \geq 0$. This corresponds to having Alice guess a $(1-\varepsilon)$-fraction of the elements in the intersection $U \cap V$.

Remark 2. The two-player communication complexity of TaskAlloc$_n$ is $O(n)$, since Alice can always just send her complete input (represented as the n-bit characteristic vector) to Bob, claim all the tasks in her input, and have Bob claim the remaining tasks. Theorem 7.3 shows that this strategy is optimal. Moreover, if we wish to find all the elements in the intersection, when the intersection is of size $\Omega(n)$, repeatedly sampling a random element is the optimal strategy up to a $\log(n)$ factor.

7.3 Lower Bound on Rooted Spanning Tree

Next we show how to apply the lower bound from the previous section to obtain a lower bound on computing a rooted spanning tree. In Section 6.1.2 we had an $\Omega(n/B)$ lower bound for diameter 2 networks, which we obtained as a corollary of the counting lower bound (Theorem 6.3). This lower bound applied only when the count was initially unknown. As we remarked in Section 6.1.2, it seems like knowing the size of the network should not make it easier to find a spanning tree; we now show that this intuition is correct.

Formally, the *distributed rooted spanning tree* problem in a network $G = (V, E)$ requires each node $v \in V$ to output a value $p_v \in V \cup \{\perp\}$, such that the edges $\{(v, p_v) \mid v \in V\}$ form a rooted spanning tree of G (oriented upwards toward the root). Exactly one node v may output $p_v = \perp$, and this node is the root of the tree. In each round of the algorithm, each node $v \in V$ broadcasts B bits, which are delivered to all of v's out-neighbors in G. Each node of G initially knows the size n of the graph and has a unique identifier (UID) drawn from the set $[n]$.

Our lower bound shows that finding a rooted spanning tree is hard even in a restricted class $\mathcal{G}_n$ of networks, where each $G \in \mathcal{G}_n$ is strongly-connected, has a diameter of 2, and has no simple directed path of length more than 4. (In particular, all spanning trees have depth at most 4, so the algorithm cannot be "confused" by long paths or by tall potential spanning trees.) The reduction is almost the same as the reduction we used to prove Theorem 6.3.

Theorem 7.4. *Any algorithm for finding a rooted spanning tree in networks of $\mathcal{G}_n$ requires at least $\Omega(n/B)$ rounds to succeed with probability $1/2$.*

Proof. We prove the theorem by reduction to the two-party task allocation problem TASKALLOC_{n-2}. Specifically, we show that if there is an algorithm for finding a rooted spanning tree in all networks of $\mathcal{G}_n$ which requires t rounds to succeed with probability $1/2$, then there is a public-coin protocol for solving TASKALLOC_{n-2} with communication complexity $O(B \cdot t)$. The theorem then follows from Theorem 7.3.

Fix an algorithm $\mathcal{A}$ for finding a rooted spanning tree. Given inputs U, V (respectively), Alice and Bob can solve TASKALLOC_{n-2} by simulating the execution of $\mathcal{A}$ in a network $G_{U,V} = ([n], E_{U,V})$, where

$$\begin{aligned} E_{U,V} = (\{n-1, n\} \times [n-2]) \cup \{(n-1, n), (n, n-1)\} \\ \cup (U \times \{n-1\}) \cup (V \times \{n\}) . \end{aligned}$$

Informally, in $G_{U,V}$ nodes $n-1$ and n represent Alice and Bob respectively, and nodes $1, \ldots, n-2$ represent the task set of the TASKALLOC_{n-2} problem. Nodes $n-1$ and n always have edges to all nodes of the network, regardless of the input. In addition, the nodes of U have edges to node $n-1$ (that is, to "Alice") and the nodes of V have edges to node n ("Bob"). It is easy to verify that $G_{U,V} \in \mathcal{G}_n$.

Alice and Bob cooperate to simulate the execution of $\mathcal{A}$, using the public randomness to assign outcomes to the coin tosses of nodes $1, \ldots, n$; bit $n \cdot k + i - 1$ of the public random string is interpreted as bit k of node i's randomness (for $i \in [n]$

and $k = 0, 1, 2, \dots$). Alice is responsible for locally simulating nodes $U \cup \{n-1\}$; she keeps track of these nodes' states throughout the execution. Similarly, Bob is responsible for simulating nodes $V \cup \{n\}$. (The nodes in $U \cap V$ are simulated by both players independently.)

To simulate one round of $\mathcal{A}$, the players update the states of their locally-simulated nodes as follows: Alice computes the messages output by nodes $U \cup \{n-1\}$, and Bob computes the messages output by nodes $V \cup \{n-1\}$ in the current round; then Alice and Bob send each other the messages output by nodes $n-1$ and n (resp.). Now Alice computes the new state of each node in U after receiving the messages sent by nodes $n-1$ and n, and Bob does the same for the nodes in V. Finally, Alice computes the new state of node $n-1$ after receiving the messages sent by nodes $U \cup \{n\}$, and Bob updates the state of node n after receiving the messages of nodes $V \cup \{n-1\}$. Note that in the final step, Alice and Bob know *which* nodes' messages to deliver, because Alice knows U and Bob knows V. It is not hard to see that for nodes $i \in U \cap V$, Alice and Bob agree on the local state of i at every step of the simulation.

Suppose that $\mathcal{A}$ succeeds with probability at least $1/2$ after t rounds. Then after simulating round t of the execution, with probability at least $1/2$ each node $i \in [n-2]$ outputs a parent $p_i \in [n] \cup \{\perp\}$, with exactly one node $r \in [n]$ outputting $\perp$. The edges $\{(i, p_i)\}$ form a directed spanning tree with root r. To handle the root r, the protocol concludes with one final exchange: Alice sends Bob one bit b indicating whether some node $i \in U$ (that Alice was simulating) output $p_i = \perp$. Finally the players output the following sets:

$$A = \{i \in U \mid p_i = n-1\} \cup \{r, \text{ if } r \in U \text{ and } p_r = \perp\};$$
$$B = \{i \in V \mid p_i = n\} \cup \{r, \text{ if } r \in V \text{ and } p_r = \perp \text{ and } b = 0\}.$$

It is easy to verify that A, B form a valid output on instance (U, V), as each node in $[n-2]$ except possibly r must choose either $n-1$ or n as its parent (but not both), and if $r \in [n-2]$ then it is assigned to exactly one player.

The total amount of communication used by the protocol is $2Bt + 1 = O(Bt)$, and a correct output is produced with probability at least $1/2$. □

The lower bound above can be shown to be nearly-tight for networks of constant diameter (and in particular, the class $\mathcal{G}_n$); more generally, in networks of diameter D it is possible to construct a rooted spanning tree in $O(D^2 + n \log n / B)$ rounds, as we will see in Section 7.6. It is also easy to show that $O(D + |E|/B)$ rounds suffice for networks of any diameter, by simply disseminating all edges using pipelining. The time complexity of finding a spanning tree in networks with diameter $D = \omega(\sqrt{n})$ and $|E| = \omega(n)$ remains open to the best of our knowledge.

7.4 Multiparty Complexity in the Shared Blackboard Model

In this section we study the complexity of distributed task allocation in the shared blackboard communication model. First note that the two-player lower bound (Theorem 7.3) can be embedded into the multi-player setting, yielding the following lower bound:

Theorem 7.5. *The shared-blackboard communication complexity of multi-player task allocation is $\Omega(n)$. Further, for any $\alpha > 0$, $m \geq \alpha n + 2$, and $n \geq 1/\alpha$, there is a class of inputs with task-player expansion α for which some player needs to communicate $\Omega(1/\alpha)$ bits.*

Proof. We can embed a 2-player task allocation instance in an m-player game by using only two of the players and giving no tasks to the rest. Therefore, it follows from Theorem 7.3 that the communication complexity of $\text{TASKALLOC}_{m,n}$ is $\Omega(n)$ for any m.

For the bound involving the expansion α, choose two players $u, v \in V$, and partition the tasks into two sets $T', T \setminus T'$, where $|T'| = 1/\alpha$. All the tasks in $T' \setminus T$ are assigned to each of the players in $V \setminus \{u, v\}$. As for the tasks in T', we use them to construct a random 2-player input for the player u and v as in Theorem 7.3. Because the sub-problem defined by $\{u, v\}$ and T' is statistically independent of the problem defined by the remaining players and tasks, u and v have to solve their $2/\alpha$-task subproblem by themselves[2]; hence, as in Theorem 7.3, either u or v has to communicate at least $\Omega(1/\alpha)$ bits.

The task-player expansion of our overall input assignment is α: for any set $S \subseteq T$ of tasks, each task $x \in S \cap T'$ is assigned to at least one player (either u or v or both), and each task $x \in S \setminus T'$ is assigned to $m - 2$ players (all players except u, v). Thus, if S is a set of tasks that includes some task from $T \setminus T'$, we have

$$\frac{|\bigcup_{x \in S} V_x|}{|S|} = \frac{m-2}{s} \geq \frac{m-2}{n} \geq \alpha.$$

If S includes only tasks from T', then its size is bounded by $|T'| = 1/\alpha$; in the worst case all tasks will be assigned to only one player (u or v), so

$$\frac{|\bigcup_{x \in S} V_x|}{|S|} \geq \frac{1}{s} \geq \alpha.$$

Therefore the task-player expansion across all subsets of tasks is at least α. □

There is a simple deterministic strategy that nearly matches the lower bound above in terms of total bits communicated: we fix some arbitrary ordering $v_1, \ldots, v_m$

[2]Indeed, since our lower bound assumes public randomness, players u and v are able to *simulate* each of the other players, as the inputs of the other players are constant.

of the players. The first player goes first and announces its input by writing it on the shared blackboard. The second player goes next, and writes on the blackboard only the tasks in its input that were not announced by the next player. The third player writes the tasks from its input that were not claimed by the first two players, and so on. Finally, after the last player's turn, each player outputs exactly the tasks that it wrote on the board. The total bit complexity is $O(n \log n)$, because each task is written on the blackboard exactly once. However, the number of rounds is m in the worst case, and messages may be as large as $\Omega(n \log n)$ if the first player has $\Omega(n)$ tasks in its input. In [41] we gave a randomized algorithm that addresses the first concern by reducing the number of rounds to $O(\log m)$, but still uses messages of size $\Omega(n \log n)$. We do not include this algorithm here. Instead, we consider the case where the input has large task-player expansion, and give a simple algorithm that solves $\textsc{TaskAlloc}_{m,n}$ quickly using messages of size $O(\log(n+m))$.

In our algorithm, at the end of each round some tasks are permanently assigned to a player. We let $T(i)$ denote the set of tasks that have not been permanently assigned by the beginning of round $i+1$ (that is, by time i), and we let $V(i)$ denote the set of players that still have some unassigned tasks at the beginning of round $i+1$. Let $H(i)$ be the subgraph of H (the task-player graph) induced by $V(i) \cup T(i)$. In a similar manner, we use $n(i) := |T(i)|$, $m(i) := |V(i)|$ and $X_v(i)$ to denote the number of remaining tasks, the number of remaining players, and player v's remaining (unassigned) tasks at the beginning of round i.

We now consider the case where in each round, each player can only send a message of at most $B = O(\log(n+m))$ bits. We give an algorithm that has a good round complexity when the expansion $\alpha(H)$ of the task-player graph is large (cf. Definition 7.1). In the following, we assume that $\alpha(H) \leq 1$, i.e., the number of players does not exceed the number of tasks. (If $\alpha(H) > 1$, all appearances of $\alpha(H)$ in our bounds can be replaced by 1.)

Description of the algorithm. In each round, every player v picks a random task $x \in X_v(i)$ uniformly, and proposes the assignment (v, x) by writing it on the blackboard. Task x is then permanently assigned to the smallest player u that attempted to claim it (i.e., the first player that wrote (u, x) on the blackboard). Unassigned tasks $y \in T(i)$ for which no assignment (v, y) was proposed in round i remain unassigned. We continue until all tasks in T have been assigned to some player.

Analysis of the running time. First, observe that in each round until the algorithm terminates, at least one task is proposed and then permanently assigned to some player. Therefore the algorithm terminates after at most n rounds, and to analyze its running time it is sufficient to consider the first n rounds.

For $1 \leq i \leq n$, we will let $\mathcal{H}_i$ denote the distribution induced by the algorithm on histories up to time i, including all random choices made in rounds $1, \ldots, i-1$. We abuse notation slightly by also using $\mathcal{H}_i$ to denote the support of this distribution, that is, the set of all possible histories up to time i.

In the rest of this section we assume a fixed input assignment H with task-player expansion $\alpha(H)$, and analyze the performance of the algorithm on H. We begin with a proof outline, which presents the main definitions and lemmas, and then proceed to fill in the technical details.

Proof outline. We show that in each round, the algorithm makes progress in one of two ways: either a large fraction of tasks become assigned, or a large fraction of players lose a large fraction of their task-player edges. We begin by making these notions of progress formal. Let $\lambda \in (0, \alpha(H)/16]$ be a parameter whose value will be fixed later.

Definition 7.6 (Task-reducing times). *Given a history $h \in \mathcal{H}_i$ of the algorithm, we say that time i is a* task-reducing time *if the expected number of tasks assigned in round $i+1$ given h is at least $\lambda|T(i)|$.*

Definition 7.7 (Edge-reducing times). *Given a history $h \in \mathcal{H}_i$ of the algorithm, time i is called* edge-reducing for player $v \in V(i)$ *if given h, in expectation at least $\left(1 - \sqrt{\lambda/\alpha(H)}\right) \cdot |X_v(i)|$ tasks from $X_v(i)$ are permanently assigned in round $i+1$. Time i is called* edge-reducing *if it is edge-reducing for at least $(1 - \sqrt{\lambda/\alpha(H)})|V(i)|$ players.*

Informally, if a time is task-reducing, we make progress because many tasks become assigned. On the other hand, if the time is not task-reducing, this means that many players picked the same task to propose (because each player proposes one task, but not many tasks were proposed in total). Each task x proposed in round $i+1$ becomes assigned to some player, and the other players v then remove this task from their remaining input $X_v(i+1)$, causing edge (v, x) to be removed from $H(i+1)$. If H has good expansion, many players are incident to (i.e., have in their input) some task among the tasks proposed in round $i+1$, and all such players now shed all edges corresponding to proposed tasks. Thus we can show:

Lemma 7.8. *For each time i and any possible history $h \in \mathcal{H}_i$, either time i is task-reducing, or time i is edge-reducing.*

To see why this is sufficient for the algorithm to terminate quickly, consider the simple case where λ and $\alpha(H)$ are both constant. Then in expectation, each task-reducing round causes a constant fraction of tasks to be eliminated, and each edge-reducing round causes a constant fraction of players to shed a constant fraction of their edges in the task-player graphs. After roughly $\log(n)$ edge-reducing rounds, a constant fraction of players have no tasks remaining, and they are removed from $V(i)$. To eliminate all players (and hence all tasks) we require logarithmically-many such "phases", so the overall time complexity is $O(\log n \cdot \log m)$.

Our definition of task-reducing and edge-reducing rounds is stated in terms of expectations, but we can use concentration of measure to show that in both cases we make progress with at least constant probability.

Definition 7.9 (Successful rounds). *If $|X_v(i+1)| \leq (\lambda/\alpha(H))^{1/4} \cdot |X_v(i)|$, then we say that round $i+1$ is a* successful edge-reducing round for player v. *We say that round $i+1$ is* successful *if either*

- $|T(i+1)| \leq (1-\lambda/2)|T(i)|$, *or*
- *There are at least* $(1-(\lambda/\alpha(H)^{1/8}))|V(i)|$ *players* v *for which round* $i+1$ *is a successful edge-reducing round.*

Lemma 7.10. *There is a constant* $q \approx 0.12$ *such that given any history* $h \in \mathcal{H}_i$, *round* $i+1$ *is successful with probability at least* q.

We now apply this result to analyze the total number of successful rounds in an execution. We have shown that each round is successful with constant probability; however, the rounds are not independent from each other. We therefore model the algorithm's progress as a submartingale. Let $S(0,i)$ be a random variable counting the number of successful rounds up to time i. We show that the sequence $Z_0, \ldots, Z_k$ defined by $Z_i := S(0,i) - qi$ is a submartingale, and an easy application of Azuma's inequality then yields the following:

Lemma 7.11. *For any* $k \geq 0$ *and constant* $\delta \in (0,1)$ *we have*

$$\Pr_{h \sim \mathcal{H}_k}\left[S(0,k) < \delta q k\right] \leq e^{-[q(1-\delta)]^2 k/2}.$$

It remains to show that when sufficiently many successful rounds occur, the algorithm terminates. Recall that a successful round may be either task-reducing or edge-reducing; we now show that for the algorithm to terminate, it is sufficient to have

$$N_T := \log_{\frac{1}{1-\frac{\lambda}{2}}} n = \frac{\log n}{\log \frac{1}{1-\frac{\lambda}{2}}}$$

successful task-reducing rounds, or

$$N_E := 5 \log_{\left(\frac{\alpha(H)}{\lambda}\right)^{1/4}}(n) \log_{\left(\frac{4}{5}\right)\left(\frac{\alpha(H)}{\lambda}\right)^{1/8}}(m) = \frac{160 \log(n) \log(m)}{\log\left(\frac{\alpha(H)}{\lambda}\right)\left(\log\left(\frac{\alpha(H)}{\lambda}\right) + \log\left(\frac{4}{5}\right)\right)}$$

successful edge-reducing rounds. Let $N = N_T + N_E$.

Lemma 7.12. *If* h *is a history that contains at least* N *successful rounds, then the algorithm terminates in* h.

Finally, combining Lemmas 7.11 and 7.12 yields the following bounds:

Theorem 7.13. *With high probability, the algorithm terminates in at most* T *rounds, where*

$$T = O\left(\frac{\log m \log n}{\log^2(\alpha(H)\log m)}\right) \qquad \text{if } \alpha(H) = \Omega\left(\frac{1}{\log m}\right), \text{ and}$$
$$T = O\left(\frac{\log n}{\alpha(H)}\right) \qquad \text{if } \alpha(H) = O\left(\frac{1}{\log m}\right).$$

Let us now give the full proofs of these results.

The technical details.

Lemma 7.8. For each time i and any possible history $h \in \mathcal{H}_i$, either time i is task-reducing, or time i is edge-reducing.

Proof. Fix a history $h \in \mathcal{H}(i)$. Let $S \subseteq T(i)$ be a random variable representing the number of tasks that are assigned in round $i+1$, given h. If time i is not task-reducing, then $\mathbb{E}[S] < \lambda|T(i)|$. We will show that in this case time i is edge-reducing for a large fraction of remaining players. In the remainder of the proof all events, probabilities and expectations are implicitly conditioned on the history h.

Let C_u be the event that in round $i+1$, player $u \in V(i)$ picks a task $x \in X_v(i)$ that is also picked by another player $v \in V(i)$. From the assumption that time i is not task-reducing,

$$\sum_{u\in V(i)} \Pr(C_u) > |V(i)| - \mathbb{E}[S] > |V(i)| - \lambda|T(i)| \geq |V(i)| \cdot \left(1 - \frac{\lambda}{\alpha(H)}\right). \qquad (7.4.1)$$

The last inequality follows because $|V(i)| \geq \alpha(H)|T(i)|$, by the assumption that H has task-player expansion $\alpha(H)$.

For $u \in V(i)$, let Z_u be the fraction of tasks in $X_u(i)$ that are assigned to some player in round $i+1$, and let Z'_u be the fraction of tasks in $X_u(i)$ that are proposed by other players $v \in V(i) \setminus \{u\}$. Clearly, $Z'_u \leq Z_u$, and therefore also $\mathbb{E}[Z'_u] \leq \mathbb{E}[Z_u]$. Also, given that $Z'_u = x$ (that is, the other players propose an x-fraction out of player u's tasks), the probability that C_u occurs (that is, u proposes a task that is also proposed by some other player) is x. We have

$$\begin{aligned}
\Pr(C_u) &= \sum_{x=1/|X_u(i)|}^{1} \Pr(Z'_u = x) \cdot \Pr(C_u | Z'_u = x) \\
&= \sum_{x=1/|X_u(i)|}^{1} \Pr(Z'_u = x) \cdot x = \mathbb{E}[Z'_u] \leq \mathbb{E}[Z_u].
\end{aligned}$$

We need to show that for at least $\left(1 - \sqrt{\lambda/\alpha(H)}\right)|V(i)|$ players $u \in V(i)$ we have $\mathbb{E}[Z_u] \geq \left(1 - \sqrt{\lambda/\alpha(H)}\right)$. (i.e., time i is edge-reducing for these players). Suppose not. Then more than a $\sqrt{\lambda/\alpha(H)}$-fraction of players have $\mathbb{E}\left[Z_u\right] < \left(1 - \sqrt{\lambda/\alpha(H)}\right)$, and the remaining players have $\mathbb{E}\left[Z_u\right] \leq 1$ (as Z_u is a fraction, its value never exceeds 1). Therefore we can write

$$\begin{aligned}
\sum_{u\in V(i)} \Pr(C_u) &\leq \sum_{u\in V(i)} \mathbb{E}[Z_u] \\
&< \left(1 - \sqrt{\frac{\lambda}{\alpha(H)}}\right) |V(i)| \cdot 1 + \sqrt{\frac{\lambda}{\alpha(H)}} \cdot |V(i)| \cdot \left(1 - \sqrt{\frac{\lambda}{\alpha(H)}}\right) \\
&= \left(1 - \frac{\lambda}{\alpha(H)}\right) \cdot |V(i)|.
\end{aligned}$$

a contradiction to Inequality (7.4.1). □

Lemma 7.10. There is a constant $q \approx 0.12$ such that given any history $h \in \mathcal{H}_i$, round $i+1$ is successful with probability at least q.

Proof. Fix $h \in \mathcal{H}_i$. In the sequel all probabilities and expectations are implicitly conditioned on h.

First, suppose that in h, time i is task-reducing. For a task $x \in T(i)$, let S_x be an indicator random variable that takes the value 1 iff task x is assigned in round $i+1$. The sum $S = \sum_{x \in T(i)} S_x$ is the number of tasks that are assigned to some player in round $i+1$.

By our assumption that time i is task-reducing, $\mathbb{E}[S] \geq \lambda |T(i)|$. To apply a concentration bound, we observe that the process by which players choose tasks to propose can be viewed as a balls-into-bins process, where each ball (player) independently chooses a random bin (task) according to some distribution (which depends on its remaining tasks). The indicator variable S_x is then 1 iff some ball (player) landed in bin (task) x. It is well known that the set of variables counting the number of balls in each bin is negatively associated, and that taking non-decreasing functions of a set of negatively-associated variables preserves negative association [42]. Therefore the set $\{S_x \mid x \in T(i)\}$ is negatively associated (it is obtained by taking the non-decreasing function that maps the number of players that chose task x to 1 iff this number is greater than 0). This allows us to apply the Chernoff-Hoeffding bound to the sum S [42], obtaining

$$\Pr\left[S \leq \frac{\lambda |T(i)|}{2}\right] \leq \Pr\left[S \leq \frac{E[S]}{2}\right] \leq e^{-(1/2^2)E[S]/2} \leq e^{-1/8} \approx 0.88. \tag{7.4.2}$$

In the last inequality we used the fact that $E[S] \geq 1$, because at least one task becomes assigned in each round until the algorithm terminates. It follows from (7.4.2) that round $i+1$ is successful with probability at least $1 - e^{-1/8} \approx 0.12$.

If time i is not task-reducing, then by Lemma 7.8 it is edge-reducing. Let $v \in V(i)$ be a player for which time i is edge-reducing, and let $Y = |X_v(i+1)|$ be the number of edges (i.e., unassigned tasks) that player v has left after round $i+1$. We are interested in the probability that round $i+1$ is *successful for player* v, that is,

$$Y \leq \left(\frac{\lambda}{\alpha(H)}\right)^{1/4} \cdot |X_v(i)|.$$

Since time i is edge-reducing for v, $\mathbb{E}[Y] \leq \sqrt{\lambda/\alpha(H)} \cdot |X_v(i)|$ (by Definition 7.7), and therefore by Markov,

$$\Pr\left[Y > \left(\frac{\lambda}{\alpha(H)}\right)^{1/4} \cdot |X_v(i)|\right] < \frac{\sqrt{\lambda/\alpha(H)} \cdot |X_v(i)|}{(\lambda/\alpha(H))^{1/4} \cdot |X_v(i)|} = \left(\frac{\lambda}{\alpha(H)}\right)^{1/4}. \tag{7.4.3}$$

This holds for any player $v \in V(i)$ for which time i is edge-reducing.

We assumed that time i is edge-reducing, that is, time i is edge-reducing for at least

$(1-\sqrt{\lambda/\alpha(H)})|V(i)|$ players. Now let Z be the number of players for which round $i+1$ is either not edge-reducing, or edge-reducing but not successful. If $Z \le (\lambda/\alpha(H))^{1/8}$, then by Definition 7.9, round $i+1$ is successful. We will use Markov's inequality to show that this holds with at least a constant probability.

From (7.4.3), linearity of expectation, and the fact that round i is edge-reducing for at least $(1-\sqrt{\lambda/\alpha(H)})|V(i)|$ players,

$$\mathbb{E}[Z] \le \sqrt{\frac{\lambda}{\alpha(H)}}|V(i)| + \left(\frac{\lambda}{\alpha(H)}\right)^{1/4} \cdot \left(1-\sqrt{\frac{\lambda}{\alpha(H)}}\right)|V(i)|$$
$$= \left(\frac{\lambda}{\alpha(H)}\right)^{1/4}\left(1+\left(\frac{\lambda}{\alpha(H)}\right)^{1/4} - \sqrt{\frac{\lambda}{\alpha(H)}}\right)|V(i)|.$$

The maximum of $f(x) = 1 + x - x^2$ is obtained at $x = 1/2$ and equals $5/4$, so

$$\mathbb{E}[Z] \le \frac{5}{4}\left(\frac{\lambda}{\alpha(H)}\right)^{1/4}|V(i)|.$$

By Markov,

$$\Pr\left[Z > \left(\frac{\lambda}{\alpha(H)}\right)^{1/8}|V(i)|\right] < \frac{\frac{5}{4}\left(\frac{\lambda}{\alpha(H)}\right)^{1/4}|V(i)|}{\left(\frac{\lambda}{\alpha(H)}\right)^{1/8}|V(i)|} = \frac{5}{4}\left(\frac{\lambda}{\alpha(H)}\right)^{1/8}$$
$$\le \frac{5}{4}\left(\frac{1}{16}\right)^{1/8} \approx 0.88.$$

Now set $q := \min\{1-e^{-1/8}, 1-(5/4)(1/16)^{1/8}\} \approx 0.12$. We have shown that if time i is task-reducing then round $i+1$ is successful with probability at least q, and otherwise time i is edge-reducing and round $i+1$ is still successful with probability at least q. This completes the proof. $\square$

In the sequel, let S be a random variable counting the number of successful rounds in the history, and let q be the constant from Lemma 7.10. Also, let $h[i,j]$ denote times $i,\ldots,j$ of h, and let $S(i,j)$ be short-hand for $S(h[i,j])$.

Lemma 7.11. For any $k \ge 0$ and constant $\delta \in (0,1)$ we have

$$\Pr_{h\sim\mathcal{H}_k}\left[S < \delta qk\right] \le e^{-[q(1-\delta)]^2 k/2}.$$

Proof. Let $Z_0,\ldots,Z_k$ be the sequence defined by

$$Z_i := S(0,i) - qi.$$

Then for any history $h \in \mathcal{H}_k$,

$$\begin{aligned}
&\mathbb{E}\left[Z_{i+1} \,\middle|\, h[0,i]\right] = \mathbb{E}\left[S(0,i+1) - qi \,\middle|\, h[0,i]\right] \\
&= \mathbb{E}\left[S(i,i+1) - q + S(0,i) - qi \,\middle|\, h[0,i]\right] \\
&= \Pr\left[S(i,i+1) = 1 \,\middle|\, h[0,i]\right] - q + Z_i \geq q - q + Z_i = Z_i.
\end{aligned}$$

(In the next-to-last step we used Lemma 7.10, which shows that for any $h' \in \mathcal{H}_i$, $\Pr[S(i,i+1) = 1 \mid h'] \geq q$.) This shows that $Z_0, \ldots, Z_k$ is a submartingale (with respect to the filtration $\{h[0,i]\}_i$ that reveals the first i rounds of the execution). Moreover, since $S(i,i+1) \in \{0,1\}$ for all i, we have

$$\begin{aligned}
&|Z_{i+1} - Z_i| = |S(0,i+1) - q(i+1) - S(0,i) + qi| = |S(i,i+1) - q| \\
&\leq \min\{q, 1-q\} < 1.
\end{aligned}$$

Thus, Azume's inequality yields

$$\Pr\left[Z_k - Z_0 < -(1-\delta)qk\right] \leq e^{-\frac{((1-\delta)qk)^2}{2\sum_{i=1}^{k} 1^2}} = e^{-[q(1-\delta)]^2 k/2}.$$

By definition of the sequence we have $Z_0 = S(0,0) - 0 = 0$ and $Z_k = S(0,k) - qk$; therefore

$$\Pr\left[S < \delta qk\right] = \Pr\left[S - qk < \delta qk - qk\right] = \Pr\left[Z_k - Z_0 < -(1-\delta)qk\right] \leq e^{-[q(1-\delta)]^2 k/2}.$$

□

Lemma 7.12. If h is a history that contains at least N successful rounds, then the algorithm terminates in h.

Proof. Since h contains at least $N = N_T + N_E$ successful rounds, it contains either N_T successful task-reducing rounds or N_E successful edge-reducing rounds. In the first case it is easy to see that the algorithm terminates, as each successful task-reducing round eliminates at least a $\lambda/2$-fraction of unassigned tasks; after $N_T = \log_{1/(1-\lambda/2)} n$ such rounds no tasks remain.

In the second case, let $\ell := \log_{(\alpha(H)/\lambda)^{1/4}} n$, and let us divide the successful edge-reducing rounds into blocks of 5ℓ rounds. Note that, since each edge-reducing round that is successful for player v eliminates at least a $(1 - (\lambda/\alpha(H))^{1/4})$-fraction of v's unassigned tasks, after ℓ such rounds player v is eliminated. We now show that each 5ℓ-round block eliminates at least a $(1 - (5/4)(\lambda/\alpha(H))^{1/8})$-fraction of remaining players.

Let $m', m'' \leq m$ be the number of players that still have unassigned tasks at the beginning and end of the block, respectively. Consider a matrix where the rows are indexed by the m' initial players in the block, the columns are indexed by round number inside the block, and entry (v,i) of the matrix is 1 if the i-th round is a successful edge-reducing round for player v or if player v has no tasks left at the beginning of the round, and 0 otherwise. Let us count the number of zeroes in the matrix.

- On one hand, each round in the block is a successful edge-reducing round, so in each column i, if k players still have unassigned tasks at time $i-1$, then there are at most $(\lambda/\alpha(H))^{1/8}k$ zeroes in the column. Since $k \le m'$ and the matrix contains 5ℓ columns, the total number of zeroes is at most $5(\lambda/\alpha(H))^{1/8}m'\ell$.
- On the other hand, each player for which there are at least ℓ successful edge-reducing rounds in the block is eliminated; since m'' players remain, there are at least m'' rows that contain less than ℓ one entries, that is, they contain at least $5\ell - \ell = 4\ell$ zeroes. The matrix thus contains at least $4\ell m''$ zeroes.

Since the number of zeroes is bounded from below by $4\ell m''$ and from above by $5(\lambda/\alpha(H))^{1/8}m'\ell$, we must have

$$4\ell m'' \le 5\left(\frac{\lambda}{\alpha(H)}\right)^{1/8} m'\ell,$$

that is,

$$m'' \le \frac{5}{4}\left(\frac{\lambda}{\alpha(H)}\right)^{1/8} m'.$$

This shows that all but a $(5/4)(\lambda/\alpha(H))^{1/8}$-fraction of players are eliminated.

We already saw in Lemma 7.10 that $(5/4)(\lambda/\alpha(H))^{1/8} \approx 0.88 < 1$, so the number of player does shrink after every block. After $\log_{(4/5)(\alpha(H)/\lambda)^{1/8}} m$ blocks, each comprising 5ℓ rounds, all players are eliminated and the algorithm completes. Since $N_E = 5\ell \cdot \log_{(4/5)(\alpha(H)/\lambda)^{1/8}} m$, N_E successful edge-reducing rounds are sufficient for termination. □

Theorem 7.13. With high probability, the algorithm terminates in at most T rounds, where

$$T = O\left(\frac{\log m \log n}{\log^2(\alpha(H)\log m)}\right) \qquad \text{if } \alpha(H) = \Omega\left(\frac{1}{\log m}\right), \text{ and}$$
$$T = O\left(\frac{\log n}{\alpha(H)}\right) \qquad \text{if } \alpha(H) = O\left(\frac{1}{\log m}\right).$$

Proof. Consider the first cN rounds, where c is a constant that will be fixed later. By Lemma 7.11, for histories of length cN,

$$\Pr[S < N] = \Pr\left[S < \frac{1}{c}\cdot cN\right] \le e^{-[q(1-1/c)]^2 \cdot cN/2}.$$

As we saw in Lemma 7.12, whenever $S \ge N$ the algorithm terminates. Therefore the algorithm succeeds within cN rounds with probability at least $1 - e^{-c'N}$, where

$$c' := \left[q\left(1-\frac{1}{c}\right)\right]^2 \cdot \frac{c}{2} \xrightarrow[c\to\infty]{} \infty$$

is a constant that can be chosen arbitrarily large by increasing c.

Recall that $N_T = O(\log n/\log(1/(1-\lambda)))$, $N_E = O(\log n \log m/\log^2(\alpha(H)/\lambda))$, and $N = N_T + N_E$. Let us now assume that the algorithm is run on a family of inputs H with $\alpha(H) \geq \alpha$ for some fixed expansion $\alpha = \alpha(m,n) \leq 1$. The only constraint on λ is $\lambda \in (0, \alpha/16]$, but since we assume that $\alpha \leq 1$ this implies $\lambda \in (0, 1/16]$. For such small values of λ we have $1 - \lambda \approx e^{-\lambda}$ and consequently

$$\log \frac{1}{1-\lambda} = -\log(1-\lambda) \approx -\log e^{-\lambda} = \Theta(\lambda).$$

Therefore $N_T = O(\log n/\lambda)$.

If $\alpha = O(1/\log m)$, we set $\lambda = \alpha/16$, so that $\alpha/\lambda = 16$. For this choice of λ we have $N = N_T + N_E = O(\log n/\alpha) + N_E = O(\log n \log m)$. Because $\alpha = O(1/\log m)$, the term $O(\log n/\alpha)$ dominates, and we have $N = O(\log n/\alpha)$.

If $\alpha = \Omega(1/\log m)$, then we choose

$$\lambda = \frac{\log(\alpha \log m)}{\alpha \log m}.$$

For this choice, since $\alpha \leq 1$,

$$N_T = O\left(\frac{\alpha(H) \log n \log m}{\log(\alpha \log m)}\right) = O\left(\frac{\log n \log m}{\log(\alpha \log m)}\right)$$

and

$$N_E = O\left(\frac{\log n \log m}{\log^2\left(\frac{\alpha \cdot \alpha \log m}{\log \alpha \log m}\right)}\right) = O\left(\frac{\log n \log m}{\log^2(\alpha \log m)}\right).$$

Therefore $N = O(\log n \log m/\log^2(\alpha \log m))$.

As for the success probability, in both cases above our choice of λ yields $N = \Omega(\sqrt{\log n \log m})$, so for any $\varepsilon(m,n) = 1/\mathrm{poly}(m,n)$, by choosing c large enough we can guarantee success with probability $1 - e^{-c'N} = 1 - \varepsilon(m,n)$. □

Remark 7.14. Since each player writes a B-bit message on the blackboard in each round, the total bit complexity of the algorithm is $mB \cdot T$, where T is the running time from Theorem 7.13. In typical scenario where $m = O(n)$, this is optimal to within polylogarithmic factors (by Theorem 7.5). However, if the number of players greatly exceeds the number of tasks, it becomes wasteful to have all the players propose task assignments in each round.

7.5 Arbitrary Networks

In previous sections we discussed task dissemination in traditional models for communication games, such as two-player and multi-player shared blackboard. We can also solve task dissemination in the more decentralized setting, where players are connected by an arbitrary communication network, using ideas introduced in previous chapters.

To emulate a shared blackboard we use pipelining: recall from Lemma 6.19 that if we can fit β tokens in each message, then we can disseminate k tokens throughout the network in $D + \lceil k/\beta \rceil$ rounds. If D is not known to the nodes in advance, then as we saw in Section 6.3, we can obtain an upper bound on it in $\tilde{O}(D + \sqrt{m/B})$ rounds, where B is the message bandwidth. Using these ingredients, we solve task allocation as follows:

I. **Diameter estimation:** the players execute the simultaneous variant of Algorithm 6.1 to solve the HF_m problem. In parallel, nodes find the minimum UID in the network, by appending to every message the smallest UID heard so far. The node with the minimum UID is elected to be a leader. It is responsible for collecting proposed task assignments from the other nodes and deciding the final disposition of each task.

   Algorithm 6.3 halts at every node within $\tilde{O}(D + \sqrt{m/B})$ rounds, and provides all nodes with an estimate $\hat{D}$ for the diameter of the network.

II. **Proposal:** nodes use pipelining to propose task assignments $(v, x) \in \bigcup_{v \in V} \{v\} \times X_v$ to the leader, as follows. For each token $x \in T$, we treat all the task-player edges $\{(v, x) \mid v \in V, x \in X_v\}$ as a *single* token: the "identifier" of the token is x, but it carries with it auxiliary information in the form of the UID of some node v that proposes to claim it. When node u receives a proposal (v, x) in a message, or if $u = v$ and task x is in player v's input, node u adds the proposal (v, x) to its sending queue iff it is the first proposal it has heard for task x. Subsequent proposals (v', x) are discarded.

   This phase lasts $\hat{D} + \lceil n/\beta \rceil$ rounds. At the end, the leader decides on final assignments for all tasks, by examining the proposals it has received and selecting for each task x one of the nodes v that proposed the assignment (v, x).

III. **Assignment:** the leader's decisions are disseminated throughout the network using pipelining. Since there are n tasks and the leader assigns each task to a unique node, we have a total of n tokens, so again $\hat{D} + \lceil n/\beta \rceil$ rounds are enough to disseminate them.

   Upon receiving the final assignments, each node outputs the set of tasks that the leader assigned to it.

It is easy to see that if we project each message (v, x) onto its second component x we obtain an execution of the pipelining algorithm from Lemma 6.19. The total number of tokens is n, the number of tasks. Consequently, in $O(\hat{D} + n/\beta)$ rounds, all nodes receive at least one proposal for each task.

Theorem 7.15. *The algorithm described above solves* $\textsc{TaskAlloc}_{m,n}$ *with high probability in time* $\tilde{O}(D + \sqrt{m/B} + n/B)$.

Remark. In the typical case, when $m = O(n)$, the time bound in the Theorem 7.15 simplifies to $\tilde{O}(D + n/B)$. The algorithm described above is randomized, because the diameter-estimation algorithm from [96] is randomized. However, when $m = O(n)$ we can replace this part by a deterministic coarse upper bound on the diameter D: simply use pipelining to count the number of players in the network (see [94]), and use this number m as an upper bound on the diameter. This yields a deterministic $O(D + n/B)$-round algorithm for task dissemination.

7.6 An Algorithm for Constructing Rooted Spanning Trees

We conclude our technical results with a simple algorithm for computing rooted spanning trees in directed broadcast networks. Our strategy is similar to the solution for task allocation in Section 7.5: we treat each network node as both a task and a player, where the input to player v is its set of in-children. Since we need to make sure that we do not create any cycles, we have to make task assignments in a more coordinated fashion than in Section 7.5. Therefore we assign children to parents in a top-down fashion, from the root towards the leaves.

We first assume that the nodes know the diameter D, or a linear upper bound on D. The first step, as in Section 7.5, is to select a leader r, which will serve as the root of the tree. Subsequently the algorithm runs in D phases. In the first phase, the root node r assigns all its in-neighbors as its children, and communicates this decision by applying the token dissemination protocol described allocation: the players are the nodes that are already assigned to some parent node and that still have unassigned in-neighbors; the tasks are all the unassigned in-neighbors of the players. Hence, the players of phase i are a subset of the nodes at in-distance $i - 1$ from the root (the ones that have in-neighbors at in-distance i from the root), and the tasks are all the nodes at in-distance i from the root. The algorithm terminates as soon as all nodes are assigned to some parent node.

Theorem 7.16. *The above algorithm solves the spanning tree problem in $O(D^2 + n\log(n)/B)$ rounds.*

Proof. It follows from the construction of the algorithm that in phase i all nodes at in-distance i from the root are assigned to a parent node. Therefore, the time complexity of the algorithm is determined through D sequential executions of the task assignment protocol from Section 7.5. Let k_i be the number of nodes at in-distance i from the root. The number of tasks in tokens in phase i is k_i. Therefore the running time of the task assignment protocol of round i is $O(D + k_i)$ (recall that we assumed that the nodes know D). Hence, the overall time complexity is

$$O\left(D^2 + \sum_{i=1}^{D} k_i \cdot \frac{\log(n)}{B}\right) = O\left(D^2 + \frac{n \log n}{B}\right).$$

□

Dealing with an unknown diameter. If the diameter is initially unknown and we plug in the time complexity from Theorem 7.15, we obtain an overall time complexity of $O\big(D(D + \sqrt{n/B}\log n) + n\log(n)/B\big)$. This can be slightly improved by observing that phases do not necessarily need to be synchronized. As soon as a node receives a notification that it has been assigned to some parent node, it can start broadcasting its in-neighbors so that they can be assigned. With this modification, the additive $\sqrt{n/B}\log n$ penalty term for an unknown diameter is paid only once, instead of D times (once per phase); the penalty is dominated by the $n\log n/B$ in Theorem 7.16, so the overall time complexity from Theorem 7.16 is preserved. Note, however, that spanning tree constructed in this way is no longer a BFS tree.

7.7 Discussion and Open Problems

Our results in this chapter leave several problems open. First, our lower bound from Section 7.3 shows that computing a spanning tree requires $\Omega(n/B)$ rounds, but the best upper bound of which we are aware, even assuming the diameter D is known in advance, is $O(\min\{D + |E|/B, D^2 + n\log n/B\})$. For dense networks with a large diameter, the bounds do not match. However, TaskAlloc$_{n,n}$ can be solved in $O(D + n\log n/B)$ rounds (see Section 7.5). The existence of a fast spanning tree algorithm implies a fast algorithm for task allocation, where we view each node as both a task and a player; the input of each player is its set of in-neighbors (viewed as "tasks"), and its output is the set of in-neighbors that chose it as their parent in the tree. The other direction is not necessarily true, since in general a task allocation may contain cycles (when we view nodes as both tasks and players). If the network is sufficiently dense, and perhaps enjoys good expansion as well, is it nevertheless possible to use a fast task allocation algorithm to find a rooted spanning tree? Can we prove that cycles are unlikely to occur, and if so, can we resolve the few cycles that do occur quickly?

Another open problem concerns task-allocation with good task-player expansion and the hardness of finding a spanning tree in directed constant-degree expanders. In a constant-degree network with bounded bandwidth B, each node only receives $O(B)$ bits of information per round. This bottleneck bounds the number of nodes with which a given node can "exchange meaningful information", even though the diameter is small. To tackle this issue in a communication-complexity setting, we could charge the protocol not just for the total bits exchanged, but also for activating the (directed) channel between two players. We could then ask what is the smallest number of channels that must be activated to solve TaskAlloc or other problems. All the algorithms we have given for TaskAlloc require either all players to exchange information with all other players (as in the shared blackboard model), or one player to exchange $\Omega(n)$ information with all other players (as in the algorithm from Section 7.5). A strong lower bound on the number of player-to-player channels that must be activated would yield insight into the problem and perhaps lead to a lower bound on finding spanning trees in directed constant-degree expanders.

Chapter 8

Conclusion

We conclude the thesis by giving a brief summary of the results, evaluating their significance and implications, and describing some open problems.

8.1 Summary of the Results

The problems we investigated in the thesis broadly fall into five classes.

8.1.1 Information Dissemination

One main contribution in Chapter 3 has been to show that all-to-all information dissemination, the "hardest" of single-shot tasks, can be solved efficiently in dynamic networks, even under minimal assumptions. We showed that we can disseminate n pieces of information in $O(n + n^2/T)$ rounds in T-interval connected graphs, and also extended this result to several weaker models, including the beep model and asynchronous wakeup.

A key information-dissemination technique introduced in Chapter 3 is *pipelining* (Section 3.4.2), which allows us to disseminate k tokens over a static backbone, with a latency of at most k for each token. Pipelining works particularly well in static graphs, and we re-used it in the second part of the thesis to solve the HF_n task (Section 6.3) and to emulate a shared-blackboard over a general network (Section 7.5).

8.1.2 Counting and Computing Functions of the Input

We studied counting and approximate counting in both parts of the thesis. In the first part, we showed that in dynamic networks, we can obtain an approximate count in nearly-linear time, using messages of only polylogarithmic size. In the second part we showed that randomized ε-approximate counting requires $\Omega(\min\{n, 1/\varepsilon^2\}/B)$ rounds, even in static networks of known diameter 2, and that when it is *not* known that the diameter is 2, even computing a constant approximation to the count requires $\Omega(\sqrt{n/B})$ rounds. This is tight to within polylogarithmic factors.

We also studied the complexity of computing duplication-insensitive functions, which are as hard as the HF_n task. Any counting or token-dissemination algorithm is also capable of computing any duplication-insensitive function (in the sense of a $(0, b)$-reduction, where b is the size of the function's input domain), so in some sense all the algorithms discussed thus far are also algorithms for computing a duplication-insensitive function; the converse is not necessarily true. In Chapter 6 we studied the hardness of solving HF_n (and therefore also of computing duplication-insensitive functions), and showed that this task requires $D + \tilde{\Theta}(\sqrt{n/B})$ rounds, even for D as small as 2.

8.1.3 Coordinated Consensus

We showed that consensus, in some sense the easiest possible global task, is as hard as the HF_n task, which implies that it is equivalent in hardness to computing duplication-insensitive functions. Using tools from epistemic logic, we studied the time required to achieve *simultaneous* consensus, and showed that this task requires n rounds in every execution; this implies that simultaneous consensus is as hard as finding an upper bound on n. Finally, as a compromise, we introduced an intermediate task, Δ-coordinated consensus, and showed that its best-case complexity is much better than that of simultaneous consensus. We also gave two lower bounds on Δ-coordinated consensus. The developments in this part of the thesis echo the theoretical development of Δ-*approximate common knowledge* [69], but this is the first instance in which this concept has found application in practice, through both our algorithms and our lower bounds.

8.1.4 Task Allocation

In Chapter 7 we introduced the *task allocation* problem, and showed that its two-player randomized communication complexity is $\Omega(n)$. We used this lower bound to obtain an $\Omega(n/B)$-round lower bound on computing a spanning tree in networks of diameter 2. We also gave two algorithms for task allocation: the first is designed for the shared-blackboard model, and performs well when the input is "well-balanced"; the second algorithm is designed for general networks and requires no assumptions on the input.

8.2 Discussion and Concluding Remarks

8.2.1 Designing Algorithms for Ad-Hoc Wireless Networks

As far as algorithm design is concerned, the main take-away from this thesis is that randomized, approximate, and topology-insensitive techniques tend to cope well with uncertainty and dynamic conditions, while more precise approaches, or approaches that involve pre-computing network infrastructure, do not scale well.

Overall Design Philosophy

Many distributed algorithms in the literature fail to find practical application because they are too *fragile*: they were developed under certain assumptions (e.g., that the communication graph is a unit-disk graph), and do not degrade gracefully when these assumptions are violated. In contrast, the algorithms we gave here were typically developed under fairly pessimistic assumptions: an adversarially-chosen dynamic network, very little initial information, and so on. However, we also built in mechanisms for exploiting better conditions where available (e.g., more stability, better expansion, etc.). Wherever possible, we designed our algorithms with a built-in *termination test*, which assessed whether conditions so far have been good enough to allow the algorithm to terminate, or whether more time is required.

Our goal in this thesis has been to design algorithms that *cope with the worst, but hope for the best.* We believe that this is the correct design philosophy under conditions as uncertain as those of ad-hoc wireless networks.

Remark 8.1. Ours is far from the first attempt to design algorithms that are *adaptive* to circumstances, but previous attempts have revealed some interesting and paradoxical behavior of existing models, which can make adaptivity hard.

A relevant comparison here is to *early-terminating consensus* in the f-crash fault model (e.g., [83]), where nodes attempt to decide before the worst-case running time of $f - 1$ rounds where possible. The study of early-terminating consensus in the f-crash fault model has exposed a curious and somewhat paradoxical feature of models that have a bounded number of faults: *the fault-free executions are exactly the executions with the worst-case running time.* This is because when no faults occur, the adversary "keeps its cards close to its chest", conserving all its power; the algorithm must tread carefully because it does not know which nodes may suddenly become faulty. In contrast, when many faults occur early on in the execution, the faulty nodes are exposed, and the algorithm can stop worrying about them. Therefore the best-behaved execution of the system is the worst-case execution for algorithms. This is arguably not a desirable feature in a fault model.

The 1-interval connected dynamic graph model is not subject to this curious best-case/worst-case inversion: in each round, the adversary receives "a fresh supply of energy", and can choose new edges independent of its past choices.

Randomization and Approximation in Distributed Algorithms

In classical sequential computing, it is believed by many that randomization does not add computation power; in particular, it is conjectured that BPP, the class of languages decided by a probabilistic bounded-error Turing machine, is equal to P. It is already famously known that in distributed computing this is not the case: randomization can allow us to solve problems that are impossible to solve deterministically (e.g., consensus [54], the best-known instance of this separation), and even when a deterministic solution is possible, randomization can provide an exponential improvement in the running time (e.g., [18, 89], where it is shown that randomized algorithms for broadcast in small-diameter networks are exponentially faster than the best deterministic algorithm).

Our work provides more examples of the power of randomization: we showed, for example, that while randomly computing a constant approximation of the count requires only $\tilde{O}(D)$ rounds in static networks, deterministic attempts to solve the same problem will run up against a $\Omega(D + n/B)$-round lower bound. Similarly, computing an approximate answer is often much easier than finding the exact answer, as our $\Omega(D + n/B)$-round lower bound on *randomized exact counting* attests.

It is interesting to ask whether and in what circumstances randomized and approximate algorithms are useful in practice; for example, computing an approximate count of the size of the network seems to be good enough for most applications, while perhaps computing an approximation of the diameter that may err on the low side is less useful.

Topology-Oblivious Approaches

The traditional approach to any sort of data aggregation is to first compute a spanning tree, then aggregate on top of the tree. However, we showed in this thesis that this strategy is not always optimal for directed networks (much less for *dynamic* networks, where any spanning tree we find may not persist into the future). We showed that even under very good conditions — a network of diameter 2, with no simple paths of length more than 4 — computing a rooted spanning tree requires $\Omega(n/B)$; in contrast, certain "easy" functions (like minimum or the approximate count) can be computed much faster, in $O(D + \sqrt{n/B})$ rounds. This means that for small-diameter networks, the "investment" in computing a spanning tree is only worth it if we intend to compute at least $\Omega(\sqrt{n/B})$ minima.

In general it seems that "topology-oblivious" algorithms, such as the algorithm in Section 6.3 and gossip algorithms [84, 114], may be better suited for directed networks. However, there is much work to be done in extending and analyzing gossip algorithms in directed and dynamic networks (see Section 8.3.2 below).

8.2.2 Lower Bound Techniques

In distributed computing as a whole, there are two diametrically-opposed models that exemplify the challenges a distributed algorithm must overcome. Each model comes with a technique for proving lower bounds, which tends to work particularly well in that model (though it may often be very useful in other contexts as well). In this thesis we have used a model that combines features of both models, and we employed both lower bound techniques. We will now make these connections concrete, and later (in Section 8.3.3) point out a direction for future work.

The LOCAL Model and Proofs by Indistinguishability

In the LOCAL model (e.g., [108, 125]), value is placed on the locality of the computation, rather than on the amount of information conveyed. The network is assumed to be some arbitrary graph, but the algorithm is only allowed to run for a small number k of rounds, which means that nodes can only communicate with their near neighbors. The amount of communication between pairs of nodes is subject to a sharp cut-off: nodes at distance no more than k from each other can exchange as much information as they like, while nodes at distance greater than k cannot communicate at all and therefore exchange zero information.

Proofs by indistinguishability have been used to great effect in the LOCAL model (e.g., [95, 98]). The key here is that since nodes only communicate with their k-neighborhood, they cannot observe any change *outside* this neighborhood, and cannot adapt their behavior to the change. Indistinguishability works well when there is *zero communication* between the part of the network where we wish to make the change and the part of the network whose behavior should (but cannot be) affected by the change. This is why indistinguishability is typically used together with *full-information protocols*, where nodes are not charged for the amount of information they send; it does not matter how much information is sent, because we are already restricted to arguing about nodes that have not communicated with each other *at all*. That is also the way in which indistinguishability was used in this thesis: in Chapter 4, we showed that even under full-information, coordinated consensus is hard; elsewhere, we used indistinguishability only to argue about time-nodes $(u, t), (v, t')$ such that $(u, t) \notin \mathsf{past}(v, t')$, that is, (u, t) has not heard from node (v, t') at all.

The CONGEST Model and Information-Theoretic Lower Bounds

Lower bounds in the CONGEST model [125] often assume that the network graph is fully connected, but the size of each message is bounded by a parameter B, which is very small compared to n.[1] This is the opposite of the LOCAL model: now all nodes can reach each other easily, but the amount of information they can exchange is bounded. To prove lower bounds in the CONGEST model, one resorts to *information-theoretic arguments*, such as the communication-complexity lower bounds used here.

Our network model is not fully-connected, so is not quite the classical CONGEST model; however, it is no coincidence that our only applications for communication-complexity lower bounds were in small-diameter networks (diameter 2, to be specific). The defining feature of the CONGEST model is that everyone *hears from* everyone quickly, but the *amount of information* they gain from the interaction is small.

We believe that there is interesting work to be done in combining the two proof techniques above. This is discussed further in Section 8.3.3.

8.3 Open Problems

8.3.1 Direct Extensions of the Results in this Thesis

Let us point out several open problems immediately arising from our results here.

[1] There is also work where the network is a general graph, e.g., [133]. In any case, in the CONGEST model one typically aims for a runtime lower bound that exceeds the network's diameter, in contrast to the LOCAL model, where no such lower bound is possible.

Information dissemination. We know that in 1-interval connected networks, there is a tight $\Theta(n^2)$-round bound on token-forwarding algorithms. The best non-token-forwarding algorithm known at present achieves only a logarithmic improvement [67], but no matching lower bound is known. What is the true gain from giving up token-forwarding, which has the advantage of great simplicity, in favor of more complicated schemes, such as network coding?

Counting vs. adaptive adversaries. Our nearly-linear approximate-counting algorithm from Section 3.6 can only cope with an *oblivious* adversary, which commits to the entire dynamic graph in advance. Is it possible to design a randomized algorithm that works against an *adaptive* adversary and improves upon the deterministic $O(n^2)$-round solution?

Deterministic HF_n in static networks. In Chapter 6 we gave an $\Omega(D + \sqrt{n/B})$-round lower bound on randomized HF_n, and a nearly-matching randomized algorithm. What is the *deterministic* round complexity of the HF_n task? Clearly the randomized lower bound carries over to deterministic algorithms, but is it still tight?

I conjecture that a stronger lower bound, $\Omega(D+n/B)$, might perhaps be possible. For technical reasons, a 2-party reduction in the style of the ones from Chapter 6 will not yield this result, as the reduction exposes too much information to the players (this argument can be made concrete). Therefore new tools would be required.

However, if one wishes to tackle deterministic algorithms for HF_n instead of (or in addition to) a stronger lower bound, our work also suggests an approach to do this. The randomized algorithm from Section 6.3 has only one probabilistic component — the token selection scheme (Section 6.3.2). Once tokens are selected at the beginning of the execution, the remainder of the algorithm is entirely deterministic. Therefore, to come up with a deterministic HF_n algorithm, one might try to find a deterministic token selection scheme that satisfies the condition (G) laid out in Section 6.3.2.

HF_n and function computation in dense networks. As we pointed out in Section 7.7, our lower bounds based on communication complexity involve networks with very bad expansion (as bad as $1/n$). It is not at all clear that these results continue to hold in denser networks that enjoy better connectivity properties. The techniques we use at present do not seem up to the challenge of resolving this question, as we discussed in Section 7.7. On the other hand, our algorithms also cannot exploit good connectivity to obtain strongly-sublinear running time: they are strongly based on pipelining, where the latency is always at least the number of tokens that must be disseminated (in our case typically $\text{polylog}(n/B)$).

Part of the problem appears to be the fact that we are using *local broadcast*; in each round, each node must send the same message to all its neighbors, which seems to inhibit the use of the multiple edge-disjoint paths offered by dense graphs with good expansion. One might consider asking: do local broadcast algorithms suffer from an inherent disadvantage compared to unicast-based algorithms when it comes to exploiting graphs with good conductance or expansion?

8.3.2 Designing Better Algorithms for Ad-Hoc Networks

Computing with a Dynamic Node Set

Throughout this thesis we have assumed that the set of participants in the computation is fixed and does not change throughout the execution. It is very interesting to consider extensions of our results to the more general setting where nodes can join or leave the network. This immediately raises some definitional questions: what does it *mean* to compute a function of the nodes' input, when the set of nodes can change over time? Should we aim for some notion of computation-in-the-limit, as in population protocols [5]? Or should we stay closer to the single-shot task framework we used here?

We suggest the following general scheme: consider an ongoing computation, where nodes continually maintain some estimate $\hat{f}$ of the value $f(V(t))$, where $V(t)$ is the set of participants at time t and f is the function we wish to compute. Fix some interval length Δ, which should be as small as possible. Then our criteria for a "correct" estimate $\hat{f}$ is that

- $\hat{f}(t)$ should reflect the input values of all nodes in $\bigcap_{t'=t-\Delta}^{t} V(t')$, that is, of participants that stuck around throughout the interval $[t-\Delta, t]$.
- $\hat{f}(t)$ should *not* reflect the input values of nodes in $\overline{\bigcup_{t'=t-\Delta^t V(t')}}$, that is, of participants that were not around at any point during the interval $[t-\Delta, t]$.

This seems like a reasonable notion which might be achievable in practice.

We leave the details of what we mean by "reflect" here vague. For example, if f is the minimum, and the input to node v is x_v, then we would like to have:

$$\min\left\{x_v \mid v \in \bigcup_{t'=t-\Delta}^{t} V(t')\right\} \leq \hat{f}(t) \leq \min\left\{x_v \mid v \in \bigcap_{t'=t-\Delta}^{t} V(t')\right\}.$$

Gossip in Directed or Dynamic Networks

We mentioned above that it is not clear whether algorithms based on local broadcast can scale gracefully with the density, conductance and expansion of the network. One class of algorithms that *do* exploit such properties extremely well is gossip algorithms [84], where in each round, each node contacts a random neighbor and communicates with that neighbor.

It is known that the running time of gossip in undirected graphs is closely tied to their conductance [64]. However, conductance is a *symmetric* measure, which considers all edges in the cut (there is no notion of "incoming" or "outgoing" edges); conductance is not defined for directed graphs, and we are not aware of a generalization that extends it to directed graphs. Gossip algorithms typically work in two modes: *pull*, where the node requests the contacted neighbor's information, and *push*, where the node sends the neighbor it contacted its own information. Any notion of "directed conductance" that we might try to define should reflect these two modes and the way in which they advance the algorithm's overall progress, and some of the basic ideas (e.g., the notion of a "balanced cut") do not appear to carry over. For this reason it appears that a significant technical effort is required to extend the results of [64] to the directed case.

More recently, it was shown in [28] that using gossip-like techniques, it is possible to emulate local broadcast efficiently even in graphs with poor conductance. This outstanding result offers the hope of extending our results in this thesis to models that do not assume local broadcast. However, [28] relies on having an undirected network even more strongly than [64], as the algorithm must explicitly track backwards across a chain of messages sent in the past. This is impossible when the network is directed, and becomes even less feasible when the network is dynamic. We believe that extending the algorithm from [28] to the directed case and to the dynamic case is a challenging and worthwhile problem.

Adapting Streaming Techniques to Dynamic Networks

In Section 3.1 we showed that the HF_n task is complete for the class of "easy" (duplication-insensitive) functions, where nodes can summarize everything they know in one value from the domain of the function (e.g., the minimum value observed so far). Can we extend this completeness result to cover a larger class of functions, by introducing approximation and some probability of error? We believe that the answer is yes; we have already seen one hint of such an extension in Section 3.6, where we gave an algorithm that solves approximate counting and HF_n together.

Techniques from the field of *streaming algorithms* [116] are likely to prove useful here. A streaming algorithm must process a long stream of data, using only a small amount of storage space to "remember" what it has seen so far. One popular way to do this is to use a *sketch* of the data: a sketch is a small-space, updatable representation of a large dataset X, from which the value of some function $f(X)$ can be approximately recovered [116]. For example, a trivial sketch for computing a minimum is to simply remember the minimum value seen so far, and update it as more data comes along. This should seem very familiar.

Sketches are usually designed to be *updatable* and *combinable*, but this is not quite enough for our purposes here. If we wish to compute a duplication-sensitive function by having all nodes send each other sketches, we must make the sketch resilient to duplication: we must somehow ensure that if a node receives sketches from its neighbors containing redundant "copies" of information from some source, its own sketch will incorporate only one "copy" of the information, that is, it will reflect only the original source. It is not clear how to do this in general, but one approach might be to approximately count the number of copies of each input using, e.g., an approach similar to Algorithm 3.8 (which is resilient to duplication). If our target function is not *highly* sensitive to the number of copies of each input, then small errors in the approximate count should not have a large effect on its value. There is some work to do in defining the class of functions for which this approach works, fleshing out the algorithm, and identifying approaches that work for other classes.

8.3.3 Improving Our Understanding of Information Theoretic Principles Underlying Distributed Computation

The Communication Cost of Function Computation

In this thesis we gave reductions in both directions between the three "lower" levels of the hierarchy: counting, computing duplication-insensitive functions, and consensus. However, we have not related any of these tasks to *information dissemination*, except in the trivial sense that disseminating all inputs allows us to compute anything.

Several of our lower bounds do show that some nodes must exchange a total of, e.g., $\Omega(n)$ bits between themselves, in the case of counting. Conceivably, we could relate counting to information-sending between two nodes using our reductions from Chapter 6. However, just because we can show that two nodes a, b must exchange a total of $\Omega(n)$ bits to solve counting, this does not mean that we can reduce the problem of having a send an arbitrary $\Omega(n)$-bit string to b to the problem of counting: we do not know the *contents* of the $\Omega(n)$ bits that are exchanged in the counting algorithm, and how they relate to the nodes' input. Therefore our reductions in Chapter 6 are just shy of giving us a reduction from information-sending to counting and HF_n.

Let us make this question very concrete, and frame it in the context of communication complexity. Following [126], let us define the *two-player mailing problem* as follows: Alice receives an input string $x \in \{0,1\}^n$, and Bob receives no input; the goal is for Bob to learn and output the string x. Then we conjecture that the mailing problem can be reduced to a small number of Set Disjointness instances, as follows:

Conjecture. *Given a* DISJ_n *protocol P that uses r bits of public randomness, there are parameters* $\ell = \text{polylog}(n), q \geq r$ *such that there exist*

- *A mapping* $f_{\mathsf{Alice}} : \{0,1\}^n \times \{0,1\}^{q \cdot \ell} \to (\{0,1\}^n)^\ell$ *that Alice can use to generate a sequence of inputs* $x_1, \ldots, x_\ell \in \{0,1\}^n$ *for P from her input string* $x \in \{0,1\}^n$ *and a shared random string* $s \in \{0,1\}^{q \cdot \ell}$,
- *A mapping* $f_{\mathsf{Bob}} : \{0,1\}^{q \cdot \ell} \to (\{0,1\}^n)^\ell$ *that Bob can use to generate a sequence* $y_1, \ldots, y_\ell \in \{0,1\}^n$ *of inputs for P from the shared random string* $s \in \{0,1\}^{q \cdot \ell}$, *and*
- *A mapping* $f_{\mathsf{out}} : (\{0,1\}^*)^\ell \times \{0,1\}^{q \cdot \ell} \to \{0,1\}^n$ *from ℓ transcripts of P and the shared random string to n-bit strings,*

such that for any input x, when Alice and Bob

1. *Use their shared randomness* $s = s_1, \ldots, s_\ell \in \{0,1\}^{q \cdot \ell}$ *(interpreted as a sequence of ℓ q-bit strings) to generate input sequences* $f_{\mathsf{Alice}}(x, s) = x_1, \ldots, x_\ell$ *and* $f_{\mathsf{Bob}}(s) = y_1, \ldots, y_\ell$.
2. *Execute the* DISJ_n *protocol P with each input pair (x_i, y_i) and the corresponding shared random substring s_i, and*
3. *Bob feeds the resulting ℓ transcripts $T_1, \ldots, T_\ell$ of P and the random string s into* f_{out} *to obtain an n-bit string* $f_{\mathsf{out}}((T_1, \ldots, T_\ell), s)$,

then with high probability over s, Bob correctly outputs Alice's original input x.

We are essentially claiming that Set Disjointness is a "universal communication primitive" (between two players), which *does not directly follow* from the lower bounds on the communication complexity of Set Disjointness; although we know that Alice and Bob must exchange $\Omega(n)$ bits to solve a $\textsc{Disj}_n$ instance, we do not know what *information* these bits convey about the inputs — for all we know, Alice and Bob could just be talking about the weather.

Communication Complexity in the Multi-Hop Setting

In Section 8.2.2 we discussed two useful but contrasting techniques for proving lower bounds: in proofs based on *indistinguishability*, we assume that it is "hard" for players to contact each other, but if two players have managed to contact each other, then they may potentially have shared with each other all the information they have about the execution; in proofs based on *information-theoretic* (or communication-complexity) arguments, we assume that the amount of information players can convey to each other is bounded, but that it is very easy for players to contact each other.

Both proof techniques have been used to great effect in proving lower bounds on distributed computation, but to our knowledge, they have never been *combined*. Indistinguishability arguments are used in multi-hop networks, or in settings (such as fault-prone asynchronous shared memory and asynchronous message-passing networks) where communication is infrequent and subject to the adversary's interference. In contrast, information-theoretic bounds are usually employed in shallow, synchronous networks, often in the fully-connected / shared-blackboard setting, where each message sent by one player is immediately received by all the other players. This avoids the need to argue about *indirectly-conveyed information*.

Imagine a tree (or worse, a DAG) of nodes, oriented upwards towards a root node u, with leaf nodes $x_1, \ldots, x_\ell$. If nodes perform a convergecast where each node sends a B-bit message up the tree, what does node u now know about the states of nodes $x_1, \ldots, x_\ell$? If node u can send B-bit "questions" to nodes $x_1, \ldots, x_\ell$ to try to instruct them on what information to send, what is the effect of this bounded feedback? We need some notion of *composing* the amount of uncertainty (entropy) that each node has about its children, together with the uncertainty induced by having the node send only B bits about its knowledge up the tree, and the bounded feedback provided by nodes "asking" their children for information. Moreover, we need to combine *absolute* uncertainty in the sense of indistinguishability — about parts of the network from which the node has not heard at all — with quantified, *partial* uncertainty, about nodes from which some bounded amount of information has been received.

To the best of our awareness, such techniques have not been developed and applied in the context of distributed computing, but the rich and deep field of information theory is sure to yield relevant insights and techniques.

Multi-Party Communication Complexity in the Context of Multicore Processors

Finally, let us conclude with another potential application of multi-party communication complexity, which is quite far from the topic of this thesis, but which we believe may be rewarding to explore.

In recent years, the data structures community has experienced a resurgence in lower bounds, due in large part to the seminal work of Mihai Pătraşcu in relating lower bounds in the cell-probe model to two-player communication complexity problems [122]. This progress has allowed a deeper understanding of the trade-offs between the space required to represent the data and the number of steps required to perform queries and update operations. In the two-party games of [122], Bob represents the data structure, and Alice represents the algorithm querying the data structure; each exchange between the players corresponds to a *cell probe*, where Alice asks Bob for the contents of some memory cell.

As computers gain more and more cores, it seems only natural to extend the study of data structures to the multi-core setting. This adds another interesting element: in addition to the space required to represent the data structure and the number of cell probes required to access it, we are now also interested in the amount of *communication* required between cores, as local operations that access the core's own memory are cheaper and faster than remote operations by orders of magnitude.[2] To address this challenge, it seems quite likely that the two-player techniques of [122] ought to be extended to the multi-player setting, with each player representing one core which both stores part of the data and also holds some of the queries. Such a study would allow us to ask questions about the trade-offs between redundancy in data representation and the amount of communication between cores, about the best ways to partition the data between cores, and so on; perhaps it might even provide insights that will be helpful in the design of future multi-core architectures.

[2]There is already great interest in the remote-memory-reference (RMR) model, which captures exactly this notion; however, to date, only "low-level" problems such as mutual exclusion (e.g., [86]) and simulating basic memory primitives (e.g., [9, 23]) have been studied in the RMR model.

Index

T-interval connectivity, **38**, 66
Δ-coordinated consensus, **120**
HF_n, **45**, 104, 140, 152–174
k-committee, **56**, 59
k-verification, **50**, 56

adaptive adversary, **32**, 87
approximate counting, **50**, 79, 147
asynchronous wakeup, **32**, 75

beep model, **78**

causal order, **39**
common knowledge, **101**, 107
communication complexity, **140**
complete tasks, **37**, 54
consensus, **36**, 99–137
coordinated consensus, **120**, 120–137
counting, **50**, 147

directed networks, 139–195
duplication-insensitive functions, **35**, 54
dynamic diameter, **41**, 126, 134
dynamic graph, **31**, 31–137

gap hamming distance, **144**, 147
gap set disjointness, **142**, 153, 157
globally-sensitive functions, **35**, 152

hearing from everyone, **45**

identical executions, **47**

knowledge, **46**, **101**, 99–137

non-trivial facts, **108**, 116

oblivious adversary, **32**, 79

past and future sets, **40**
pipelining, **63**, 159

reduction, **36**, 50, 56

sensitive functions, **35**, 152
set disjointness, **141**, 153, 156
similarity graph, **101**
simultaneous HF_n, **170**, 193
simultaneous consensus, **107**, 116
simultaneous tasks, **35**, 107
single-valued tasks, **35**
spanning tree, **152**, 181, 194

task allocation, **177**, 175–195
task-player expansion, **178**, 183, 184
task-player graph, **177**
tasks, **34**
termination test, **55**, 80
timing constraints, **34**
token dissemination, **50**, 55, 63
token-forwarding algorithms, **86**
two-player communication games, **140**

vertex growth, **42**, 66

Bibliography

[1] J. Aas. Understanding the Linux 2.6.8.1 CPU scheduler. Unpublished manuscript, 2005.

[2] Y. Afek, N. Alon, Z. Bar-Joseph, A. Cornejo, B. Haeupler, and F. Kuhn. Beeping a maximal independent set. In *Proceedings of the 25th international conference on Distributed computing (DISC)*, pages 32–50, 2011.

[3] 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.

[4] 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.

[5] 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.

[6] D. Angluin, J. Aspnes, D. Eisenstat, and E. Ruppert. The computational power of population protocols. *Distributed Computing*, 20(4):279–304, 2007.

[7] S. Arora and B. Barak. *Computational Complexity - A Modern Approach*. Cambridge University Press, 2009.

[8] 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.

[9] H. Attiya, D. Hendler, and P. Woelfel. Tight RMR lower bounds for mutual exclusion and other problems. In *Proc. of the 40th Annual ACM Symp. on Theory of Computing (STOC)*, pages 217–226, 2008.

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

[11] C. Avin, M. Koucký, and Z. Lotker. How to explore a fast-changing world (cover time of a simple random walk on evolving graphs). In *Proc. of 35th Coll. on Automata, Languages and Programming (ICALP)*, pages 121–132, 2008.

[12] B. Awerbuch. Optimal distributed algorithms for minimum weight spanning tree, counting, leader election and related problems (detailed summary). In *Proc. 19th Symp. on Theory of Computing (STOC)*, pages 230–240, 1987.

[13] B. Awerbuch, S. Kutten, and D. Peleg. Competitive distributed job scheduling. In *Proc. 24th Symp. on Theory of Computing (STOC)*, pages 571–580, 1992.

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

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

[16] 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.

[17] A. Bar-Noy and D. Dolev. Consensus algorithms with one-bit messages. *Distributed Computing*, 4:105–110, 1991.

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

[19] L. Barriére, P. Fraigniaud, and L. Narayanan. Robust position-based routing in wireless ad hoc networks with unstable transmission ranges. In *Proc. of 5th International Workshop on Discrete Algorithms and Methods for Mobile Computing and Communications (DIALM)*, pages 19–27, 2001.

[20] Y. Bartal and A. Rosen. The distributed k-server problem—a competitive distributed translator for k-server algorithms. In *Proc. of the 33rd Annual Symp. on Foundations of Computer Science (FOCS)*, pages 344–353, 1992.

[21] 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.

[22] C. Bettstetter, G. Resta, and P. Santi. The node distribution of the random waypoint mobility model for wireless ad hoc networks. *IEEE Transactions on Mobile Computing*, 2:257–269, 2003.

[23] V. Bhatt and P. Jayanti. Constant rmr solutions to reader writer synchronization. In *Proc. of the 29th ACM SIGACT-SIGOPS Symp. on Principles of distributed computing (PODC)*, pages 468–477, 2010.

[24] E. Bortnikov, M. Gurevich, I. Keidar, G. Kliot, and A. Shraer. Brahms: Byzantine resilient random membership sampling. *Computer Networks*, 53(13):2340–2359, 2009.

[25] P. Bose, P. Morin, I. Stojmenović, and J. Urrutia. Routing with guaranteed delivery in ad hoc wireless networks. In *Proc. of 3rd International Workshop on Discrete Algorithms and Methods for Mobile Computing and Communications (DIALM)*, pages 48–55, 1999.

[26] J. Brody, A. Chakrabarti, O. Regev, T. Vidick, and R. de Wolf. Better gap-hamming lower bounds via better round elimination. In *APPROX-RANDOM*, pages 476–489, 2010.

[27] T. Camp, J. Boleng, and V. Davies. A survey of mobility models for ad hoc network research. *Wireless Communication and Mobile Computing*, 2(5):483–502, 2002.

[28] K. Censor-Hillel, B. Haeupler, J. A. Kelner, and P. Maymounkov. Global computation in a poorly connected world: fast rumor spreading with no dependence on conductance. In *Proc. of the 44th Symp. on Theory of Computing Conference (STOC)*, pages 961–970, 2012.

[29] A. Chakrabarti and O. Regev. An optimal lower bound on the communication complexity of gap-hamming-distance. In *Proc. 43rd ACM Symp. on Theory of Computing (STOC)*, pages 51–60, 2011.

[30] F. Chierichetti, S. Lattanzi, and A. Panconesi. Rumour spreading and graph conductance. In *Proc. of the Twenty-First Annual ACM-SIAM Symp. on Discrete Algorithms (SODA)*, pages 1657–1663, 2010.

[31] B. N. Clark, C. J. Colbourn, and D. S. Johnson. Unit disk graphs. *Discrete Mathematics*, 86(1–3):165–177, 1990.

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

[33] A. Clementi, A. Monti, F. Pasquale, and R. Silvestri. Broadcasting in dynamic radio networks. *Journal of Computer and System Sciences*, 75(4):213–230, 2009.

[34] A. Clementi, A. Monti, and R. Silvestri. Fast flooding over Manhattan. In *Proc. of 29th ACM Symp. on Principles of Distributed Computing (PODC)*, pages 375–383, 2010.

[35] A. Clementi, F. Pasquale, A. Monti, and R. Silvestri. Information spreading in stationary Markovian evolving graphs. In *Proc. of IEEE Symp. on Parallel & Distributed Processing (IPDPS)*, 2009.

[36] 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.

[37] 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.

[38] M. L. Dertouzos and A. K. Mok. Multiprocessor on-line scheduling of hard-real-time tasks. *IEEE Trans. Software Eng.*, 15(12):1497–1506, 1989.

[39] D. Dolev, C. Dwork, O. Waarts, and M. Yung. Perfectly secure message transmission. *J. ACM*, 40(1):17–47, 1993.

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

[41] A. Drucker, F. Kuhn, and R. Oshman. The communication complexity of distributed task allocation. In *Proc. of the 31st Annual ACM SIGACT-SIGOPS Symp. on Principles of Distributed Computing*, 2012. To appear.

[42] D. Dubhashi and D. Ranjan. Balls and bins: A study in negative dependence. *Random Structures and Algorithms*, 13:99–124, 1996.

[43] C. Dutta, G. Pandurangan, R. Rajaraman, and Z. Sun. Information spreading in dynamic networks. *CoRR*, abs/1112.0384, 2011.

[44] C. Dwork, N. Lynch, and L. Stockmeyer. Consensus in the presence of partial synchrony. *J. ACM*, 35(2):288–323, Apr. 1988.

[45] C. Dwork and Y. Moses. Knowledge and common knowledge in a Byzantine environment: crash failures. *Information and Computation*, 88(2):156–186, 1990.

[46] C. Dwork, D. Peleg, N. N Pippenger, and E. Upfal. Fault tolerance in networks of bounded degree. In *Proc. 8th Symp. on Theory of Computing (STOC)*, pages 370–379, 1986.

[47] M. Elkin. A faster distributed protocol for constructing a minimum spanning tree. *J. Comput. Syst. Sci.*, 72(8):1282–1308, 2006.

[48] R. Fagin, J. Y. Halpern, Y. Moses, and M. Y. Vardi. *Reasoning about Knowledge*. MIT Press, Cambridge, MA, 2003.

[49] A. Ferreira. Building a reference combinatorial model for MANETs. *IEEE Network Magazine*, 18(5):24–29, 2004.

[50] A. Ferreira, A. Goldman, and J. Monteiro. On the Evaluation of Shortest Journeys in Dynamic Networks. In *Proc. of 6th IEEE Int. Symp. on Network Computing and Applications (NCA)*, pages 3–10, 2007.

[51] A. Ferreira, A. Goldman, and J. Monteiro. Performance evaluation of routing protocols for MANETs with known connectivity patterns using evolving graphs. *Wireless Networks*, 16(3):627–640, 2009.

[52] A. Fiat, Y. Rabani, and Y. Ravid. Competitive k-server algorithms. In *Journal of Computer and System Sciences*, pages 454–463, 1990.

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

[54] M. J. Fischer, N. A. Lynch, and M. Paterson. Impossibility of distributed consensus with one faulty process. In *Proc. of the 2nd Second ACM SIGACT-SIGMOD Symposium on Principles of Database Systems (PODS)*, pages 1–7, 1983.

[55] P. Frankl and V. Rödl. Forbidden intersections. *Transactions of the American Mathematical Society*, 300(1):259–286, 1987.

[56] G. N. Frederickson and N. A. Lynch. The impact of synchronous communication on the problem of electing a leader in a ring. In *Proc. 16th ACM Symp. on Theory of Computing (STOC)*, pages 493–503, 1984.

[57] M. Frigo, C. E. Leiserson, and K. H. Randall. The implementation of the cilk-5 multithreaded language. *SIGPLAN Not.*, 33:212–223, 1998.

[58] S. Frischknecht, S. Holzer, and R. Wattenhofer. Networks cannot compute their diameter in sublinear time. In *Proc. 22nd Symp. on Discrete Algorithms (SODA)*, pages 1150–1162, 2012.

[59] 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.

[60] R. G. Gallager, P. A. Humblet, and P. M. Spira. A distributed algorithm for minimum-weight spanning trees. *ACM Trans. Program. Lang. Syst.*, 5(1):66–77, Jan. 1983.

[61] A. J. Ganesh, A.-M. Kermarrec, and L. Massoulié. SCAMP: Peer-to-peer lightweight membership service for large-scale group communication. *Networked Group Communication*, pages 44–55, 2001.

[62] J. A. Garay, S. Kutten, and D. Peleg. A sublinear time distributed algorithm for minimum-weight spanning trees. *SIAM J. Comput.*, 27(1):302–316, 1998.

[63] J. A. Garay and R. Ostrovsky. Almost-everywhere secure computation. In *Proc. 27th EUROCRYPT*, pages 307–323, 2008.

[64] G. Giakkoupis. Tight bounds for rumor spreading in graphs of a given conductance. In *28th International Sym. on Theoretical Aspects of Computer Science (STACS)*, volume 9, pages 57–68, 2011.

[65] A. Gronemeier. Asymptotically optimal lower bounds on the nih-multi-party information complexity of the and-function and disjointness. In *Proc. of the 26th International Symp. on Theoretical Aspects of Computer Science (STACS)*, pages 505–516, 2009.

[66] M. Gurevich and I. Keidar. Correctness of gossip-based membership under message loss. *SIAM J. of Computing*, 39(8):3830–3859, 2010.

[67] B. Haeupler and D. R. Karger. Faster information dissemination in dynamic networks via network coding. In *Proc. of the 30th Annual ACM Symp. on Principles of Distributed Computing (PODC)*, pages 381–390, 2011.

[68] B. Haeupler and F. Kuhn. Lower bounds on information dissemination in dynamic networks. Unpublished manuscript, 2012.

[69] J. Y. Halpern and Y. Moses. Knowledge and common knowledge in a distributed environment. *J. ACM*, 37(3):549–587, 1990.

[70] J. Håstad and A. Wigderson. The randomized communication complexity of set disjointness. *Theory of Computing*, 3(1):211–219, 2007.

[71] 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.

[72] M. Herlihy and N. Shavit. The topological structure of asynchronous computability. *J. ACM*, 46(6):858–923, Nov. 1999.

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

[74] P. Indyk and D. Woodruff. Tight lower bounds for the distinct elements problem. In *Proc. 44th IEEE Symp. on Foundations of Computer Science (FOCS)*, pages 283 – 288, oct. 2003.

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

[76] S. Janson, T. Luczak, and A. Rucinski. *Random graphs*. Wiley-Interscience Series in Discrete Mathematics and Optimization. Wiley-Interscience, New York, 2000.

[77] T. S. Jayram. Hellinger strikes back: A note on the multi-party information complexity of AND. In *Proc. 13th Workshop on Randomization and Computation (RANDOM)*, pages 562–573, 2009.

[78] T. S. Jayram. Information complexity: a tutorial. In *Proc. of the 29th ACM SIGMOD-SIGACT-SIGART Symp. on Principles of Database Systems (PODS)*, pages 159–168, 2010.

[79] D. B. Johnson and D. A. Maltz. Dynamic source routing in ad hoc wireless networks. In *Mobile Computing*, chapter 5, pages 153–181. Kluwer Academic, 1996.

[80] B. Kalyanasundaram and G. Schnitger. The probabilistic communication complexity of set intersection. *SIAM J. Discrete Math.*, 5(4):545–557, 1992.

[81] M. Karchmer and A. Wigderson. Monotone circuits for connectivity require super-logarithmic depth. *SIAM J. Discrete Math.*, 3(2):255–265, 1990.

[82] 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.

[83] I. Keidar and S. Rajsbaum. On the cost of fault-tolerant consensus when there are no faults: preliminary version. *SIGACT News*, 32(2):45–63, 2001.

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

[85] 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.

[86] Y.-J. Kim and J. H. Anderson. Adaptive mutual exclusion with local spinning. *Distributed Computing*, 19(3):197–236, 2007.

[87] Y.-B. Ko and N. H. Vaidya. Location-aided routing (lar) in mobile ad hoc networks. In *Proc. of 4th Annual ACM/IEEE International Conference on Mobile Computing and Networking (MOBICOM)*, pages 66–75, 1998.

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

[89] 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.

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

[91] F. Kuhn, T. Locher, and S. Schmid. Distributed computation of the mode. In *Proc. 27th ACM Symp. on Principles of Distributed Computing (PODC)*, pages 15–24, 2008.

[92] F. Kuhn, T. Locher, and R. Wattenhofer. Tight bounds for distributed selection. In *Proc. 19th ACM Symp. on Parallelism in Algorithms and Architectures (SPAA)*, pages 145–153, 2007.

[93] F. Kuhn, N. A. Lynch, and C. C. Newport. The abstract MAC layer. *Distributed Computing*, 24(3-4):187–206, 2011.

[94] F. Kuhn, N. A. Lynch, and R. Oshman. Distributed computation in dynamic networks. In *Prof. of the 42nd Symp. on Theory of Computing (STOC)*, pages 513–522, 2010.

[95] F. Kuhn, T. Moscibroda, and R. Wattenhofer. What cannot be computed locally! In *Proc. of the 23rd Annual ACM Symp. on Principles of Distributed Computing (PODC)*, pages 300–309, 2004.

[96] F. Kuhn and R. Oshman. The complexity of data aggregation in directed networks. In *Proc. of 25th Symp. on Distributed Computing (DISC)*, pages 416–431, 2011.

[97] 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.

[98] F. Kuhn and R. Wattenhofer. On the complexity of distributed graph coloring. In *Proc. of the 25th Annual ACM Symp. on Principles of Distributed Computing (PODC)*, pages 7–15, 2006.

[99] F. Kuhn, R. Wattenhofer, Y. Zhang, and A. Zollinger. Geometric ad-hoc routing: of theory and practice. In *Proc. of the 22nd ACM Symp. on Principles of Distributed Computing (PODC)*, pages 63–72, 2003.

[100] F. Kuhn, R. Wattenhofer, and A. Zollinger. Ad-hoc networks beyond unit disk graphs. In *Proc. of 2003 Joint Workshop on Foundations of Mobile Computing (DIALM-POMC)*, pages 69–78, 2003.

[101] E. Kushilevitz and N. Nisan. *Communication complexity*. Cambridge University Press, 1997.

[102] E. Kushilevitz and E. Weinreb. The communication complexity of set-disjointness with small sets and 0-1 intersection. *Proc. of 50th Annual IEEE Symp. on Foundations of Computer Science (FOCS)*, 0:63–72, 2009.

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

[104] L. Lamport and M. Fischer. Byzantine generals and transaction commit protocols. Technical report, SRI Technical Report, Op. 62, 1982.

[105] J.-Y. Le Boudec and M. Vojnovic. The random trip model: Stability, stationarity regime, and perfect simulation. *IEEE/ACM Transactions on Networking*, 16(6):1153–1166, 2006.

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

[107] X. Li, J. Misra, and C. G. Plaxton. Maintaining the ranch topology. *J. of Parallel and Distributed Computing*, 70(11):1142–1158, 2010.

[108] N. Linial. Locality in distributed graph algorithms. *SIAM J. Comput.*, 21(1):193–201, 1992.

[109] C. L. Liu and J. W. Layland. Scheduling algorithms for multiprogramming in a hard-real-time environment. *J. ACM*, 20(1):46–61, 1973.

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

[111] N. Malpani, J. L. Welch, and N. Vaidya. Leader election algorithms for mobile ad hoc networks. In *Proc. of 4th International Workshop on Discrete Algorithms and Methods for Mobile Computing and Communications (DIALM)*, pages 96–103, 2000.

[112] Y. Moses. Optimum simultaneous consensus for general omissions is equivalent to an np oracle. In *Proc. of the 23rd International Symp. on Distributed Computing (DISC)*, pages 436–448, 2009.

[113] Y. Moses and M. R. Tuttle. Programming simultaneous actions using common knowledge. *Algorithmica*, 3:121–169, 1988.

[114] D. Mosk-Aoyama and D. Shah. Fast distributed algorithms for computing separable functions. *IEEE Transactions on Information Theory*, 54(7):2997–3007, 2008.

[115] A. Mostéfaoui, S. Rajsbaum, and M. Raynal. Conditions on input vectors for consensus solvability in asynchronous distributed systems. *J. ACM*, 50(6):922–954, 2003.

[116] S. Muthukrishnan. Data streams: Algorithms and applications. *Foundations and Trends in Theoretical Computer Science*, 1(2), 2005.

[117] A. Negro, N. Santoro, and J. Urrutia. Efficient distributed selection with bounded messages. *IEEE Trans. Parallel and Distributed Systems*, 8(4):397–401, 1997.

[118] I. Newman. Private vs. common random bits in communication complexity. *Inf. Process. Lett.*, 39:67–71, July 1991.

[119] 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.

[120] 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.

[121] A. Olszewski, J. Wolenski, and R. Janusz. *Church's Thesis After 70 Years*. Ontos Verlag, 2007.

[122] M. Patrascu. Unifying the landscape of cell-probe lower bounds. *SIAM J. Comput.*, 40(3):827–847, 2011.

[123] B. Patt-Shamir. A note on efficient aggregate queries in sensor networks. *Theor. Comput. Sci.*, 370(1-3):254–264, 2007.

[124] M. Pease, R. Shostak, and L. Lamport. Reaching agreement in the presence of faults. *J. ACM*, 27(2):228–234, 1980.

[125] D. Peleg. *Distributed Computing: A Locality-Sensitive Approach*. SIAM, 2000.

[126] D. Peleg and V. Rubinovich. A near-tight lower bound on the time complexity of distributed minimum-weight spanning tree construction. *SIAM J. Comput.*, 30(5):1427–1442, 2000.

[127] C. Plaxton, R. Rajaraman, and A. W. Richa. Accessing nearby copies of replicated objects in a distributed environment. In *Proc. of 9th ACM Symp. on Parallel Algorithms and Architectures (SPAA)*, pages 311–320, 1997.

[128] A. A. Razborov. On the distributional complexity of disjointness. *Theor. Comput. Sci.*, 106:385–390, December 1992.

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

[130] N. Santoro, M. Scheutzow, and J. B. Sidney. On the expected complexity of distributed selection. *J. Parallel and Distributed Computing*, 5(2):194–203, 1988.

[131] N. Santoro, J. B. Sidney, and S. J. Sidney. A distributed selection algorithm and its expected communication complexity. *Theoretical Computer Science*, 100(1):185–204, 1992.

[132] N. Santoro and P. Widmayer. Time is not a healer. In *Proc. 6th Symp. on Theoretical Aspects of Computer Science (STACS)*, pages 304–313. Springer-Verlag, 1989.

[133] A. D. Sarma, S. Holzer, L. Kor, A. Korman, D. Nanongkai, G. Pandurangan, D. Peleg, and R. Wattenhofer. Distributed verification and hardness of distributed approximation. In *Proc. of the 43rd ACM Symp. on Theory of Computing (STOC)*, pages 363–372, 2011.

[134] A. Schiper. Early consensus in an asynchronous system with a weak failure detector. *Distributed Computing*, 10(3):149–157, 1997.

[135] A. A. Sherstov. The multiparty communication complexity of set disjointness. *Electronic Colloquium on Computational Complexity (ECCC)*, 18:145, 2011.

[136] L. Shrira, N. Francez, and M. Rodeh. Distributed k-selection: From a sequential to a distributed algorithm. In *Proc. 2nd ACM Symp. on Principles of Distributed Computing (PODC)*, pages 143–153, 1983.

[137] A. A. Shvartsman and C. Georgiou. *Cooperative Task-Oriented Computing: Algorithms and Complexity*. Morgan&Claypool Publishers, 2011.

[138] I. Stoica, R. Morris, D. Karger, F. Kaashoek, and H. Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet applications. In *Proc. ACM SIGCOMM Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*, 2001.

[139] D. M. Topkis. Concurrent broadcast for information dissemination. *IEEE Trans. Softw. Eng.*, 11:1107–1112, 1985.

[140] T. Vidick. A concentration inequality for the overlap of a vector on a large set, with application to the communication complexity of the gap-hamming-distance problem. Electronic Colloquium on Computational Complexity (ECCC), Report TR11-051, 2010.

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

[142] D. Woodruff. Optimal space lower bounds for all frequency moments. In *Proc. of 15th Annual ACM-SIAM Symp. on Discrete Algorithms (SODA)*, pages 167–175, 2004.

[143] A. C.-C. Yao. Some complexity questions related to distributive computing(preliminary report). In *Proc. of the 11th Symp. on Theory of Computing (STOC)*, pages 209–213, 1979.