# Distributed Half-Integral Matching and Beyond[*]

Sameep Dahal and Jukka Suomela
Aalto University

**Abstract**

By prior work, it is known that any distributed graph algorithm that finds a *maximal matching* requires $\Omega(\log^* n)$ communication rounds, while it is possible to find a *maximal fractional matching* in $O(1)$ rounds in bounded-degree graphs. However, all prior $O(1)$-round algorithms for maximal fractional matching use arbitrarily fine-grained fractional values. In particular, none of them is able to find a *half-integral* solution, using only values from $\{0, \frac{1}{2}, 1\}$. We show that the use of fine-grained fractional values is necessary, and moreover we give a *complete characterization* on exactly how small values are needed: if we consider maximal fractional matching in graphs of maximum degree $\Delta = 2d$, and any distributed graph algorithm with round complexity $T(\Delta)$ that only depends on $\Delta$ and is independent of $n$, we show that the algorithm has to use fractional values with a denominator at least $2^d$. We give a new algorithm that shows that this is also sufficient.

## 1 Introduction

By prior work, it is known that there is a distributed graph algorithm that finds a *maximal fractional matching* (see Section 1.2) in $O(\Delta)$ rounds in graphs of maximum degree $\Delta$ [3]; in particular, the running time is independent of $n$ and only depends on $\Delta$. However, the algorithm uses very fine-grained fractional values; when $\Delta$ increases, the denominators grow exponentially fast. In this work we show that this is necessary: any distributed graph algorithm that finds a maximal fractional matching in $T(\Delta)$ rounds, independently of $n$, has to use fractional values with a denominator at least $2^d$ (and this is tight). In particular, there cannot be a $T(\Delta)$-rounds algorithm for finding a maximal *half-integral* matching.

### 1.1 Distributed maximal matching is hard

Maximal matching is one of the classic problems in the field of distributed graph algorithms, studied extensively since the very early days of the field in the 1980s [4, 6–8, 11, 13, 17]. In the maximal matching problem, the task is to find a matching (a set of edges without common vertices) that is not a strict subset of another matching. This is something one can trivially find in a centralized setting (pick independent edges greedily until you are stuck), but this is a challenging coordination task in a distributed setting, for two reasons:

[*]This work was supported in part by the Academy of Finland, Grant 333837.

1. One has to *break symmetry.* For example, if the input graph is a cycle, one has to select some but not all edges—the input is symmetric, but the output is not. The task is not solvable at all without resorting to, e.g., unique identifiers or randomness, and even then we cannot solve the task in constant number of rounds; maximal matching in cycles requires $\Omega(\log^* n)$ rounds [14, 16].

2. One has to solve a *local coordination* task. Even if we have a $\Delta$-regular bipartite graph, with the bipartition given, we still need $\Omega(\Delta)$ rounds to find a maximal matching, at least in sufficiently large graphs [4].

On the positive side, $O(\Delta + \log^* n)$-round distributed algorithms for finding a maximal matching in a graph of maximum degree $\Delta$ are known [17]; one can also make different trade-offs between dependency on $\Delta$ vs. $n$ [6–8], but it is impossible to achieve a running time of $T(\Delta)$, independent of $n$ [14, 16]. All of these results hold in the usual LOCAL model of distributed computing (see Section 2.2 for the details).

## 1.2 Distributed fractional matching is easier

A matching $M \subseteq E$ in a graph $G = (V, E)$ can be interpreted as a function $x$ that assigns value $x(e) = 1$ to each edge $e \in M$. If we let

$$x[v] = \sum_{e \in E : v \in e} x(e)$$

denote the sum of labels on edges incident to node $v \in V$, then we can define that function $x \colon E \to \{0, 1\}$ is a *matching* if $x[v] \leq 1$ for all $v \in V$. Moreover, $x$ is a *maximal matching* if for each edge $\{u, v\} \in E$ at least one endpoint is *saturated*, i.e., $x[u] = 1$ or $x[v] = 1$. Finally, $x$ is a *maximum matching* if it maximizes $\sum_e x(e)$.

Now we can now also consider the *fractional relaxation* of this integer program. We say that $x \colon E \to [0, 1]$ is a *fractional matching* if it satisfies $x[v] \leq 1$ for each $v \in V$, it is a *maximal fractional matching* if $x[u] = 1$ or $x[v] = 1$ for each edge $\{u, v\} \in E$, and it is a *maximum fractional matching* if it maximizes $\sum_e x(e)$. See Fig. 1 for illustrations.

Note that any maximal matching is also a maximal fractional matching, but the converse is not necessarily true. However, maximal fractional matchings share many useful properties of maximal matchings. For example, the set of saturated nodes forms a 2-approximation of a minimum vertex cover [5].

When we consider distributed graph algorithms for maximal fractional matchings, one of the obstacles discussed in Section 1.1 goes away: *we do not need to break symmetry.* For example, if the graph is a cycle, we can simply label all edges with $1/2$. More generally, if we have a $d$-regular graph, we can label all edges with $1/d$. The lower bound of $\Omega(\log^* n)$ from [14, 16] for symmetry-breaking problems no longer applies.

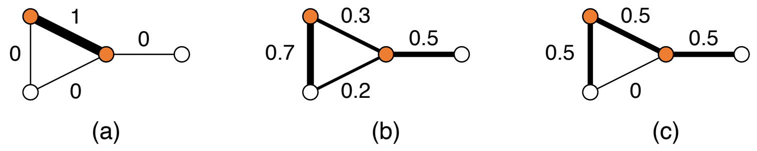

Figure 1: (a) A maximal matching. (b) A maximal fractional matching. (c) A maximal half-integral matching. The orange nodes are saturated.

While the case of non-regular graphs is much more challenging, it is nevertheless possible to design distributed algorithms that find a maximal fractional matching in $O(\Delta)$ rounds, independently of $n$ [3]. It is also known that the local coordination challenge does not disappear; $o(\Delta)$-round algorithms do not exist [10].

## 1.3 What about half-integral matchings?

The fractional matching polytope is *half-integral* (see e.g. [19, Section 30.3]). That is, there exists a maximum fractional matching in which $x(e) \in \{0, \frac{1}{2}, 1\}$ for every edge $e \in E$.

There is also a simple distributed strategy that at first seems to lead to half-integral solutions (see e.g. [2]). First, construct the *bipartite double cover* $G' = (V', E')$ of the graph $G = (V, E)$: for each node $v \in V$ we have two nodes $v_1$ and $v_2$ in $V'$, and for each edge $\{u, v\} \in E$ we have two edges $\{u_1, v_2\}$ and $\{u_2, v_1\}$ in $E'$. Now $G'$ is bipartite, and we know the bipartition, with nodes $v_1$ on one side and nodes $v_2$ on the other side. We can now apply any algorithm that finds a matching $x'$ in the bipartite graph $G'$, and this can be mapped into a half-integral matching $x$ by setting

$$x[\{u, v\}] = \frac{x'[\{u_1, v_2\}] + x'[\{u_2, v_1\}]}{2}. \tag{1}$$

Hence, we could use any distributed algorithm designed for bipartite graphs—there is a very simple algorithm that finds a maximal matching in bipartite graphs in $O(\Delta)$ rounds independently of $n$. Then by applying (1) we could turn it into a fractional matching.

There is, unfortunately, a catch: while (1) will preserve feasibility (given a matching $x'$ it will result in a fractional matching $x$), it will not preserve maximality: even if $x'$ is a maximal matching, it is not necessarily the case that $x$ is a maximal fractional matching. Could we nevertheless find a half-integral matching efficiently with a distributed algorithm?

If we consider prior distributed algorithms for maximal fractional matching [2, 3], they are very far from being able to produce half-integral matchings. For example, [2] uses fractional values with denominators as large as $2^{\Delta-1}$ and [3] is even worse. In this work we show that denominators exponential in $\Delta$ are necessary, but we can still do better than prior work.

## 1.4 Contributions

Our main result is a full characterization of exactly how fine-grained fractional values are necessary:

**Theorem 1.1** (Upper bound)**.** *There is a $T(\Delta)$-round distributed algorithm that finds a maximal fractional matching in graphs of maximum degree $\Delta \le 2d+1$ using only fractional numbers of the form $a/b$ where $a = 0, 1, \dots, 2^d$ and $b = 2^d$.*

**Theorem 1.2** (Lower bound)**.** *There is no $T(\Delta)$-round distributed algorithm for any function $T$ that finds a maximal fractional matching in graphs of maximum degree $\Delta \le 2d+2$ using only fractional numbers of the form $a/b$ where $a = 0, 1, \dots, 2^d$ and $b = 1, 2, \dots, 2^d$.*

We emphasize that the upper bound only uses multiples of $1/2^d$, while the lower bound also excludes the possibility of finding a maximal matching using, e.g., values that are multiples of $1/\Delta$.

As a corollary of these results, we also have a full characterization of the complexity of half-integral matchings:

**Corollary 1.3.** *It is possible to find a maximal half-integral matching in graphs of maximum degree $\Delta = 3$ in $O(1)$ rounds.*

**Corollary 1.4.** *It is not possible to find a maximal half-integral matching in graphs of maximum degree $\Delta = 4$ in $O(1)$ rounds.*

For larger values of $\Delta$, the range of fractional numbers we use is much smaller than in prior work. In our algorithm, the denominators is approximately $2^{\Delta/2}$, while in prior work [2] it is approximately $2^{\Delta}$.

## 1.5 Key new ideas

While the upper bound Theorem 1.1 is a relatively simple adaptation of ideas from prior work, the lower bound Theorem 1.2 requires a development of a new proof strategy.

Prior lower-bound techniques in this area tend to fall in one of these categories, each unsuitable for us:

1. The lower-bound construction is a regular graph [4, 12]. In a $\Delta$-regular graphs we can trivially find a fractional matching using the value $1/\Delta$, which is exponentially far from the lower bound in Theorem 1.2 that we aim at proving.

2. The lower-bound result aims at establishing that one needs some specific number of rounds, e.g., $\Omega(\Delta)$ rounds [4, 10, 12]. However, in Theorem 1.2 we aim at proving that even if the round complexity is, say, exponential in $\Delta$, one cannot avoid using fine-grained fractional values.

Our proof strategy superficially resembles the one used in [10, 12] in the sense that we start with one node and $k$ self-loops, which represents the local view of a node in the middle of a regular graph, and then we start unfolding the loops. At each point of the process we see what is the output the algorithm commits to, and then we continue the process until we are left with a concrete lower-bound graph. However, there are major differences; see Fig. 2:

- In [10, 12] they start with a *pair of nodes*. The nodes have self-loops, and each self-loop represents an *undirected edge*; the entire argument relies on the fact that an algorithm cannot break symmetry between two ends of such an edge. At each step they unfold a relevant loop, doubling the number of nodes, and then they mix elements from two instances, resulting in another pair of instances. In each iteration they lose one self-loop but force the algorithm to look one step further.

- In this work we start with a *single node*. The node has self-loops, but this time each self-loop represents a *long directed path*; our argument relies on the fact that an algorithm cannot break local symmetry between two nodes near the middle of the path. At each step we unfold a relevant loop, but this will turn one node into a directed path of length $\Theta(T)$. We are interested in the behavior of the algorithm both in the middle of the path and at the endpoints of the path. In each iteration we lose one self-loop but force the algorithm to use at least twice as large denominators.

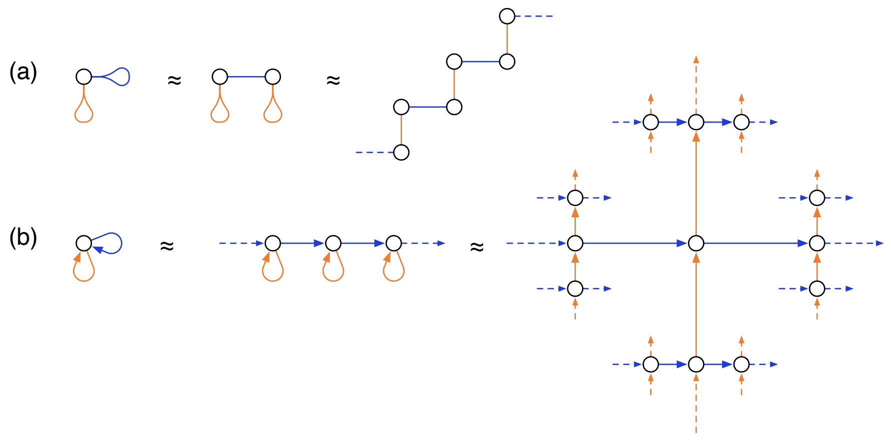

Figure 2: (a) In prior work [10,12], all the heavy lifting is done in a so-called EC model, in which edges are undirected but colored. Self-loops represent undirected edges. For example, a node with 2 self-loops represents a node in the middle of a 2-regular tree, i.e., a long path. (b) In this work, we work in the PO model. Self-loops represent long directed paths. For example, a node with 2 self-loops represents a node in the middle of a 4-regular tree in which all nodes have indegree 2 and outdegree 2.

## 2 Preliminaries

### 2.1 Graphs and self-loops

For a graph $G = (V, E)$, we write $\Delta(G)$ to denote the maximum degree of the graph. We use just $\Delta$ when $G$ is clear from the context. For any natural number $d \in \mathbb{N}$, we use $\mathcal{G}_d$ to represent the family of graphs such that $G \in \mathcal{G}_d$ if $\Delta(G) \leq d$. Throughout this work, we will assume that the maximum degree of the input graph $G$ is a globally known constant.

In what follows, we will refer to a self-loop simply as a loop. Each loop counts as one incoming and one outgoing edge (in particular, in $G_{2d}$ a node can have at most $d$ self-loops). We call a graph *loopy* if each vertex of the graph has at least one loop.

### 2.2 Model of computing

Our main results Theorems 1.1 and 1.2 hold in the usual LOCAL model [14,18]. For simplicity, we will focus here is on deterministic algorithms (even though the results are not hard to extend to randomized algorithms).

However, to prove the lower bound result, it will be convenient to first prove the lower bound in a weaker model (called PO here, following [10]) and then extend the result from the PO model to the LOCAL model. It will be easiest to define everything we need by starting with the deterministic port-numbering model (PN).

**PN model (port numbering) [1,20].** Let $G = (V, E)$ be the input graph. In the PN model, each node $v \in V$ is a computer and each edge $\{u, v\} \in E$ is a communication link between two computers. Initially, each computer is only aware of its degree; nodes of the same degree start with the same initial state.

The endpoints of the edges are labeled with *port numbers*; a node of degree $d$ can refer to its incident edges with the numbers $1, 2, \ldots, d$; see Fig. 3. The port numbering comes from an adversary; a distributed algorithm in the PN model has to work correctly for any given port numbering.

Computation proceeds in *synchronous communication rounds*. In each round, each node can

1. send a message to each neighbor,
2. receive a message from each neighbor, and
3. update its local state based on the current state and the messages it received.

After each round, each node can decide whether it stops and announces its own part of the output—in the case of the maximal fractional problem, the output of a node indicates the fractional value assigned to each incident edge. The *running time* of the algorithm is the number of rounds until all nodes have stopped and announced their local outputs.

**PO model (port numbering and orientation) [10, 15].** Algorithms in the PO model behave in exactly the same way as in the PN model. However, there is one additional piece of information available to the algorithm: each edge $\{u, v\} \in E$ is oriented (arbitrarily, by the adversary); see Fig. 3. More precisely, each node knows for each incident edge whether it is "outgoing" or "incoming".

While an arbitrary orientation may not seem particularly useful, note that the PO model is strictly stronger than the PN model. For example, if we have a graph $G$ with two nodes and one edge, it is trivial to find a proper 2-coloring of $G$ in the PO model in 0 rounds, while it is impossible to solve in the PN model in any number of rounds.

**LOCAL model [14, 18].** Algorithms in the LOCAL model also behave in exactly the same way as in the PN model, but there is again one additional piece of information available to the algorithm: each node is labeled (arbitrarily, by the adversary) with a *unique identifier* from a polynomially-sized set; see Fig. 3.

Again, LOCAL model is strictly stronger than the PO model. For example, maximal matching cannot be found in the PO model if the input graph is a cycle that is consistently oriented, while the task is solvable in the LOCAL model in $O(\log^* n)$ rounds.

However, it turns out that *constant-time* algorithms in the LOCAL model are not much stronger than algorithms in the PO model, see e.g. [9, 10]. This is the idea we will also make use of in this work: our main goal is to prove a lower bound in the LOCAL model, but it will be convenient to first study the PO model.

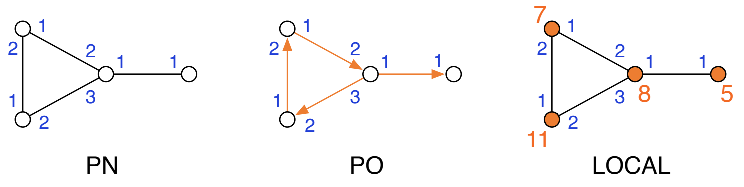

Figure 3: Models of computing used in this work.

## 2.3 Applying PO algorithms to loopy graphs

To prove the lower-bound result Theorem 1.2, we will study the output of a PO algorithm $\mathcal{A}$ in some loopy graph $G$. However, when we consider distributed graph algorithms, we usually assume that the input graph is loop-free.

However, the output of $\mathcal{A}$ in loopy graphs is nevertheless well-defined. When we refer to the output of $\mathcal{A}$ on some edge $e$ in $G$, we refer to the result of the following thought experiment: Unfold all loops in $G$, as shown in Fig. 2b, and hence we arrive at a tree $G'$. Then apply $\mathcal{A}$ in $G'$ (as the running time of $\mathcal{A}$ is independent of the size of the input graph, this is well-defined). Edge $e$ in $G$ corresponds to infinitely many edges $e'$ in $G'$, but each such edge is symmetric and hence the output of $\mathcal{A}$ on each such edge $e'$ is the same; hence we can take any such edge $e'$ and interpret its label as the output of $\mathcal{A}$ on $e$.

In particular, if $\mathcal{A}$ finds a maximal fractional matching in any loop-free graph $G'$, it will also produce a maximal fractional matching in the loopy graph $G$ (the label of the loop is counted twice).

## 3 Lower bound result

In this section we prove the lower-bound result, Theorem 1.2. It turns out that the critical resource is the number of factors of 2 in the denominators. We start by defining sets of rational numbers that will precisely capture how fine-grained values are needed.

### 3.1 Sets of rational numbers

Any natural number $x \geq 1$ can be written as $x = 2^n \cdot m$ where $n \geq 0$ and $m \equiv 1 \mod 2$. We refer to $e(x) = 2^n$ as the *even part* of $x$ and $o(x) = m$ as the *odd part* of $x$. For $x = 0$, we define $e(x) = 0$ and $o(x) = 1$.

We extend this notion to rational numbers as follows. If $x = p/q$ in the reduced form, we define the *even part of the denominator* $\bar{e}(x) = e(q)$ and the *odd part of the denominator* $\bar{o}(x) = o(q)$. For example, $\bar{e}(0/1) = \bar{e}(1/1) = 1$, $\bar{e}(1/3) = 1$ and $\bar{e}(1/4) = 4$.

For each $n \geq 1$, we define

$$\begin{aligned} R_n &= \bigl\{x \in \mathbb{Q} : 0 \leq x \leq 1 \text{ and } \bar{e}(x) = 2^n\bigr\}, \\ R_{\leq n} &= R_0 \cup R_1 \cup \cdots \cup R_n, \\ R_{\geq n} &= R_n \cup R_{n+1} \cup \cdots, \\ R_{>n} &= R_{n+1} \cup R_{n+2} \cup \cdots. \end{aligned}$$

For example, we have

$$\begin{aligned} R_0 &= \bigl\{0, 1, \tfrac{1}{3}, \tfrac{2}{3}, \tfrac{1}{5}, \tfrac{2}{5}, \tfrac{3}{5}, \tfrac{4}{5}, \dots\bigr\}, \\ R_1 &= \bigl\{\tfrac{1}{2}, \tfrac{1}{6}, \tfrac{5}{6}, \dots\bigr\}, \\ R_2 &= \bigl\{\tfrac{1}{4}, \tfrac{3}{4}, \tfrac{1}{12}, \tfrac{5}{12}, \tfrac{7}{12}, \tfrac{11}{12}, \dots\bigr\}. \end{aligned}$$

Note that for each rational number $x \in [0, 1]$ there exists exactly one $n$ such that $x \in R_n$. For $m < n$, we have $R_{\leq m} \subsetneq R_{\leq n}$.

### 3.2 High-level plan

In Section 3.3 we prove the following lemma, which essentially shows that we can without loss of generality focus on the PO model:

**Lemma 3.1.** *If there exists a $t$-time algorithm that solves the maximal fractional matching problem using values in a set $\mathscr{R}$ in the LOCAL model on any graph with maximum degree*

*$\Delta$, then there exists a $t$-time algorithm that solves the maximal fractional matching problem using values in set $\mathscr{R}$ in the PO model for any loopy graph $G$ with maximum degree $\Delta$.*

Then in Section 3.4 we prove the following lemma, which captures exactly how fine-grained rational values are needed in the PN model:

**Lemma 3.2.** *Fix natural number $d \in \mathbb{N}$. Then, for any natural number $T \in \mathbb{N}$, there does not exist any algorithm in the PO model that uses $T$ rounds and computes a valid solution for the maximal fractional matching problem using the values from $R_{\leq(d-1)}$ for loopy graphs in graph family $\mathcal{G}_{2d}$.*

By putting together Lemma 3.1 and Lemma 3.2, we obtain:

**Lemma 3.3.** *Fix a natural number $d \in \mathbb{N}$. Then, for any natural number $T \in \mathbb{N}$, there does not exist any algorithm in the LOCAL model that uses $T$ rounds and computes a valid solution for the maximal fractional matching problem using the values from $R_{\leq(d-1)}$ for the graph family $\mathcal{G}_{2d}$.*

Now Theorem 1.2 directly follows from Lemma 3.3.

## 3.3 Proof of Lemma 3.1

In [10], a similar result is shown with the exception that the edge labels are arbitrary. However, the same proof follows when we add the restriction that the edge labels come from $\mathscr{R}$. This result is a simple extension of [10, Sections 5.3–5.4], where we can see that the simulation argument does not make changes in the value used for the PO model.

## 3.4 Proof of Lemma 3.2

**Preliminary Observations.** We first make a few observations regarding our problem. First recall the way in which we use loops to represent a node in the middle of a directed path (Fig. 2).

**Observation 3.4.** *If a node has a loop then it must be saturated.*

*Proof.* If a node with a loop was not saturated, we would have a directed path of unsaturated nodes and, in particular, edges with unsaturated endpoints. □

In a saturated node, the labels of incident edges have to sum up to 1. The following observation captures a key property related to how the even parts of the denominators behave when rational numbers sum up to 1.

**Observation 3.5.** *Let $n \geq 1$ and $\frac{k}{m \cdot 2^n} \in R_n$. Consider the equation*

$$2\ell_1 + \ldots + 2\ell_r + x_1 + \ldots + x_{r'} + \frac{k}{m \cdot 2^n} = 1,$$

*where each $\ell_i$ and $x_i$ can be any non-negative rational number. Then, either $\ell_i \in R_{>n}$ or $x_i \in R_{\geq n}$ for some $i$. Put otherwise, either some $\ell_i$ has the even part of denominator larger than $2^n$ or some $x_i$ has the even part of denominator at least $2^n$.*

*Proof.* First consider the equation

$$x_1 + \ldots + x_q + \frac{k}{m \cdot 2^n} = 1$$

in which each $x_i$ can be any non-negative rational number. We show that there exists an index $i$ for which $x_i \in R_{\geq n}$. We can rewrite it as solving the equation

$$x_1 + \ldots + x_q = \frac{m \cdot 2^n - k}{m \cdot 2^n},$$

where $\frac{m \cdot 2^n - k}{m \cdot 2^n} \in R_n$. If each $x_i$ had the even part of the denominator less than $2^n$, then $x_1 + \ldots + x_q$ would also have the even part of the denominator less than $2^n$. This is because when we add two rationals $\frac{a_1}{b_1}$ and $\frac{a_2}{b_2}$ we get

$$\frac{a_1}{b_1} + \frac{a_2}{b_2} = \frac{a_1 \cdot (\ell/b_1) + a_2 \cdot (\ell/b_2)}{\ell}$$

where $\ell = \operatorname{lcm}(b_1, b_2)$, the least common multiple of $b_1$ and $b_2$. The even part of $\ell$ will be bounded above by the maximum of the even parts of $b_1$ and $b_2$. However, if $x_1 + \ldots + x_q$ has the even part of the denominator less than $2^n$, then it contradicts the fact that the sum equals $\frac{m \cdot 2^n - k}{m \cdot 2^n}$.

Now, in order to prove the original statement of Observation 3.5, it is sufficient to replace $x_{r'+i}$ by $2\ell_i$. If $x_{r'+i} \in R_{\geq n}$ then $\ell_i \in R_{>n}$. □

**Assumptions.** We now proceed to prove Lemma 3.2 by contradiction. For the sake of contradiction we assume that for any natural number $d \in \mathbb{N}$, there exists a natural number $T \in \mathbb{N}$ such that the following holds: there exists a PO algorithm $\mathcal{A}$ that solves the maximal fractional matching solution in $T$ rounds using values from the set $R_{\leq(d-1)}$ for graph family $\mathcal{G}_{2d}$.

**Properties.** Now, our lower bound construction observes the behavior of $\mathcal{A}$ on different kinds of graphs in $\mathcal{G}_{2d}$ to reason about the set of values that is used. We will construct a sequence of loopy graphs $G_0, G_1, \ldots, G_{d-1}$ to argue that the further we go, the more fine-grained value must be used by our algorithm.

For each $i = 0, 1, \ldots d-1$, we will maintain the following properties:

**P1** $G_i \in \mathcal{G}_{2d}$.

**P2** Graph $G_i$ without loops forms a tree.

**P3** Each node of $G_i$ has at least $d - i$ loops.

**P4** There is an integer $j(i) > i$ and a node $v_i$ in $G_i$ such that $\mathcal{A}$ labels at least one loop of $v_i$ with a rational value $x \in R_{j(i)}$.

**Base Case.** Our first graph $G_0$ consists of a single node $v_0$ with $d$ oriented self loops (see Fig. 4).

Graph $G_0$ satisfies properties **P1**, **P2** and **P3** by construction, so we now need to verify only **P4**. Consider that $\mathcal{A}$ assigns values $a_1, \ldots, a_d$ to the loops of $v_0$. Since $v_0$ has loops, it must be saturated (recall Observation 3.4), and hence it must satisfy that $2a_1 + 2a_2 + \ldots + 2a_d = 1$. This is equivalent to solving $a_1 + a_2 + \ldots + a_d = 1/2$ and by Observation 3.5 we know that there exists an $i$ with $a_i \in R_{\geq 1}$.

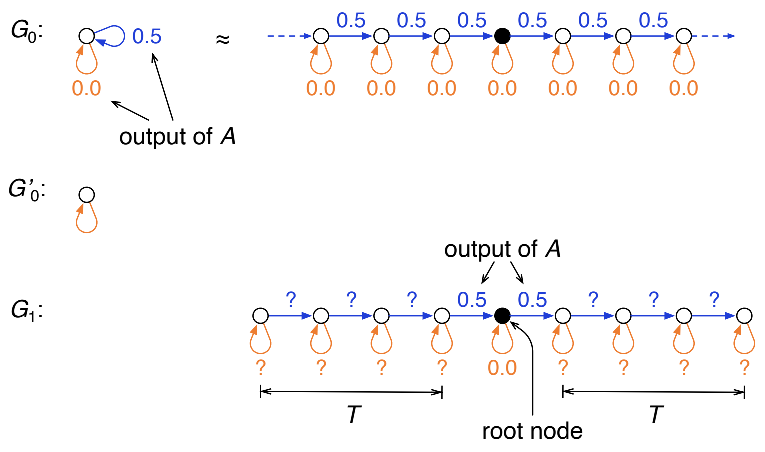

Figure 4: Construction for $d = 2$ and $T = 3$. Graph $G_0$ consists of $d$ self-loops. When we apply $\mathcal{A}$ to $G_0$, at least one of the loops will get labeled by a value in $R_{\geq 1}$; in this example the value was $0.5 \in R_1$. To construct $G_1$, we remove this loop to arrive at graph $G'_0$, take $2T + 3$ copies of $G'_0$, and connect them with a directed path. The key observation is that given the output of $\mathcal{A}$ in $G_0$ we also know the output of $\mathcal{A}$ around the node in the middle of $G_1$—this node is called the *root node* of $G_1$.

**Inductive Step.** Given $G_{i-1}$, we construct $G_i$ as follows; see Fig. 4:

**S1** Construct the graph $G'_{i-1}$ from $G_{i-1}$ by removing the loop of $v_{i-1}$ for which $\mathcal{A}$ assigned a value in $R_{j(i-1)}$.

**S2** Create $2T + 3$ copies of $G'_{i-1}$.

**S3** For each $k = 1, 2, \ldots, 2T + 2$, connect node $v_{i-1}$ in copy number $k$ to node $v_{i-1}$ in copy number $k + 1$; these new edges are called *path edges*.

**S4** Node $v_{i-1}$ in copy number $T + 2$ is called the *root node* of $G_i$.

This way we form a directed path of length $2T + 3$, with the root node in the middle of the path, as shown in Fig. 4. The key observation is that the output of algorithm $\mathcal{A}$ on the root node of $G_i$ is the same as the output of $\mathcal{A}$ for $v_{i-1}$ in $G_{i-1}$, due to the fact that the radius-$T$ neighborhood of the root node in $G_i$ is isomorphic to the radius-$T$ neighborhoods of $v_{i-1}$ in $G_{i-1}$ (once we conceptually unfold all loops). This property is illustrated in Fig. 4: compare the radius-$T$ neighborhood of the black node in the unfolding of $G_0$ with the radius-$T$ neighborhoods of the root node of $G_1$.

Given $G_{i-1}$ satisfies all the properties, we need to show that the same is true for $G_i$. **P1**, **P2** and **P3** are satisfied by construction. To prove **P4**, consider the root node of $G_i$. Since its behavior is completely characterized, we know that it will label the incident path edges with values from $R_{j(i-1)}$.

Recall that by **P2** graph $G_i$ without loops forms a tree. We will navigate in this tree, starting from the root node, and moving away from it until we satisfy **P4**. We maintain the following invariant; see Fig. 5:

**Definition 3.6** (path invariant)**.** If $v$ is the current node, and $P$ is the unique path from $v$ to the root, we have already concluded that $\mathcal{A}$ labels each edge of $P$ with a value from $R_{\geq j(i-1)}$.

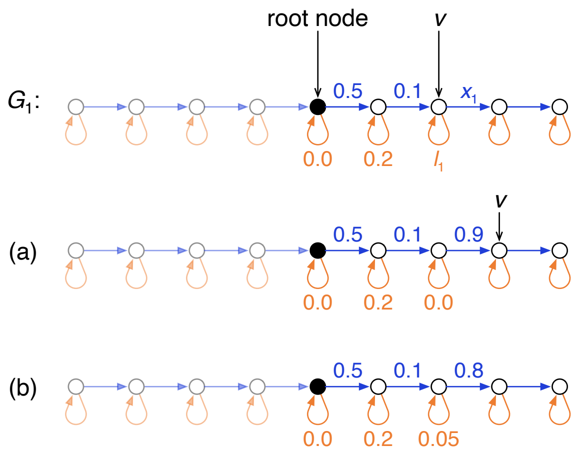

Figure 5: Inductive step in the proof of Lemma 3.2 (Section 3.4). We have already concluded that all edges in the path between $v$ and the root node are labeled with values from $R_{\geq 1}$. We now ask how algorithm $\mathcal{A}$ will label the other edges around $v$. (a) One possible solution: edge $x_1$ is labeled with a value $0.9 \in R_1$. We did not yet establish property **P4**, but we can extend the $R_1$-labeled path further away from the root node—eventually we will encounter a leaf node. (b) Another possible solution: we managed to label $x_1$ with a less fine-grained value $0.8 \in R_0$. However, this means that loop $\ell_1$ is labeled with a more fain-grained value $0.05 \in R_2$. We have established **P4**.

To get started, let $e$ be one of the path edges incident to the root node, and let $v$ be the other end of $e$. As we discussed earlier, we know that $e$ is labeled with a value from $R_{j(i-1)}$.

Now assume that we have reached some node $v$ this way. Let $P$ be the path from $v$ to the root, and let $e$ be the first edge of $P$, let $L$ be the set of loops incident to $v$, and let $X$ be the set of non-loop edges incident to $v$ that are different from $e$. That is, we already know the label of edge $e$, but we do not yet know how $\mathcal{A}$ will label $L$ and $X$.

Node $v$ is loopy, so it must be saturated. The saturation condition for $v$ is equivalent to solving the equation

$$2\ell_1 + \ldots + 2\ell_r + x_1 + \ldots + x_{r'} + \frac{k}{m \cdot 2^n} = 1,$$

where $n \geq j(i-1)$, values $\ell_i$ represent the values assigned to the loops in $L$, values $x_i$ represent the values assigned to the edges in $X$, and $\frac{k}{m \cdot 2^n}$ refers to the value from $R_{\geq j(i-1)}$ assigned to edge $e$. With the help of Observation 3.5, we know that one of the two cases must be true:

1. One of the loops in $L$ has the even part of denominator $2^{n'}$ for $n' > n$. In this case, we have established **P4**.

2. One of the edges $\{u, v\} \in X$ has the even part of denominator at least $2^n$. We have found another edge labeled with a value from $R_{\geq j(i-1)}$, and we can extend the path $P$ by moving from $v$ to $u$, still satisfying the path invariant.

Note that this process will eventually terminate, as $G_i$ without loops is a (finite) tree, and hence we will eventually reach a leaf node with $X = \emptyset$. We have established that our construction of graph $G_i$ satisfies properties **P1**–**P4**.

**Conclusion.** When we take $i = d - 1$, we have a graph $G_{d-1} \in \mathcal{G}_{2d}$ which needs to use even part of the denominator at least $2^d$. However, values with denominator $2^d$ are not present in the set $R_{\leq(d-1)}$. Thus, we have our desired contradiction.

This concludes the proof of Lemma 3.2, and hence also the proofs of Lemma 3.3 and our main lower bound result Theorem 1.2.

## 4 Upper bound result

Here, we prove the statement of Theorem 1.1. We will use the notation

$$S(d) = \left\{ \frac{i}{2^d} : i \in \{0, 1, \dots, 2^d\} \right\}.$$

We need to show that there is a $T(\Delta)$-round, independent of $n$, distributed algorithm that solves maximal fractional matching in graph family $\mathcal{G}_{2d+1}$ using labels from $S(d)$. We prove the claim by inductions, as follows:

- Base case (Lemma 4.1): $S(1)$ suffices for $\mathcal{G}_2$.
- Odd step (Lemma 4.2): if $S(d)$ suffices for $\mathcal{G}_{2d}$, then $S(d)$ also suffices for $\mathcal{G}_{2d+1}$.
- Even step (Lemma 4.3): if $S(d)$ suffices for $\mathcal{G}_{2d+1}$, then $S(d+1)$ suffices for $\mathcal{G}_{2d+2}$.

We use $T(\Delta)$ to represent the number of rounds taken by our algorithm for graph family $\mathcal{G}_\Delta$. We show that in each of the above steps, $T(\Delta)$ is just a function of $\Delta$ and is independent of number of nodes $n$. We will give a PN algorithm, which implies the existence of a LOCAL algorithm.

**Lemma 4.1.** *There is a constant-time PN algorithm that finds a maximal fractional matching in $\mathcal{G}_2$ using values from $S(1)$.*

*Proof.* In this case, we want to pick $x(e) \in \{0, \frac{1}{2}, 1\}$ for each $e \in E$. We can achieve a simple distributed algorithm with 1 round of communication. Each vertex $v$, communicates its degree to its neighbors. Any degree 2 vertex can safely assign the value $\frac{1}{2}$ to both of its incident edges. For a degree 1 vertex, it will assign the value $\frac{1}{2}$ to the incident edge if the other endpoint has degree 2 and will assign the value 1, if the other endpoint is 1 as well.

We can see that for each vertex $v$, the sum of the values assigned to its incident edges is at most 1. By the nature of our algorithm, every degree 2 node is saturated. So, every edge which has a degree 2 endpoint satisfy that one of its endpoint is saturated. The only remaining scenario is when both of the endpoints are degree 1. In this setting, our algorithm assigns the edge with value 1 in which case both of its endpoints are saturated as well. Using 1 round of communication, we have obtained a solution for the maximal fraction matching using values $\{0, \frac{1}{2}, 1\}$ when $\Delta = 2$. This gives us $T(2) = 1$. □

**Lemma 4.2.** *Fix $d \in \mathbb{N}$. Assuming that $S(d)$ is sufficient to obtain the solution for $\mathcal{G}_{2d}$, $S(d)$ is sufficient to obtain the solution for $\mathcal{G}_{2d+1}$ as well.*

*Proof.* Assume that $\mathcal{A}$ is a PN algorithm that computes the solution for $\mathcal{G}_{2d}$ using values in $S(d)$. We now describe PN algorithm $\mathcal{A}'$ that computes the solution for $G \in \mathcal{G}_{2d+1}$ using values in $S(d)$. Algorithm $\mathcal{A}'$ takes the following steps (see Fig. 6 for an illustration):

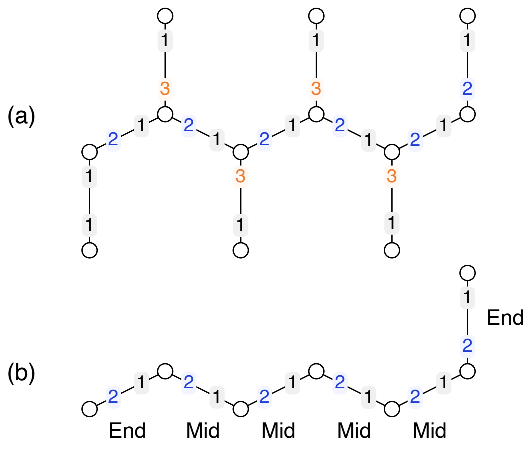

Figure 6: (a) A graph $G \in \mathcal{G}_3$, with a port numbering. (b) The subgraph $G_\ell$ for label $\ell = \{1, 2\}$, with the edge types “End” and “Mid” indicated.

**Step 1: Edge labelling.** First, we use the port numbers to define a *label* for each edge. For each edge $e = \{u, v\}$, there exists numbers $i, j \le \Delta(G)$ such that port $i$ of $u$ is connected to port $j$ of $v$. We label this edge with the set $\{i, j\}$. Then $L = \{\{i, j\} : 1 \le i, j \le \Delta(G)\}$ denotes the set of possible edge labels. We have $|L| = O(\Delta^2)$ different edge labels. For each $\ell \in L$, we define the subgraph $G_\ell$ of $G$ that contains all the edges labelled $\ell$. We write $\deg_{G_\ell}(v)$ for the degree of node $v$ in graph $G_\ell$. A key observation is that for each $\ell$ and $v$, we have $\deg_{G_\ell}(v) \le 2$, i.e., each $G_\ell$ is a collection of paths and cycles.

**Step 2: Edge Classification.** We classify each edge into two *types*: “Mid” and “End”. Consider any edge $e = \{u, v\}$ and say it had label $\ell \in L$. We say that $e$ is of type “Mid” if $\deg_{G_\ell}(u) = 2$ and $\deg_{G_\ell}(v) = 2$. Put otherwise, all edges that are in the middle of the path or part of a cycle in $G_\ell$ are classified with type “Mid”. All other edges are classified as “End”. Note that each node can determine the types of its incident edges in two rounds of communication.

**Step 3: Solve for “Mid” edges.** Consider subgraph $G'$ of $G$ that contains all edges of type “Mid”. We argue that $\Delta(G') \le 2d$. To see this, consider any vertex $v \in V$. If $\deg_G(v) = 2d + 1$, there exists $\ell \in L$ such that $\deg_{G_\ell}(v) = 1$, and therefore at least one edge adjacent to $v$ will receive type “End” and not part of $G'$. Now we have a subgraph $G'$ of $G$ with $\Delta(G') \le 2d$, and we can simulate $\mathcal{A}$ in $G'$.

**Step 4: Extend for “End” edges.** We notice that each edge $e \in G'$ satisfies the maximality condition, i.e., at least one endpoint is saturated. Thus, we now need to ensure the same for edges of type “End”. For a label $\ell \in L$, let $G_\ell^{\text{End}}$ to be set of edges labelled $\ell$ of type “End”. We know that edges of type “End” can only be part of paths of length 1 and 2 in $G_\ell^{\text{End}}$. We proceed to satisfy the maximality condition for edges of type “End” by considering them sequentially on the labels $\ell \in L$. Consider an edge $e = \{u, v\} \in G_\ell^{\text{End}}$. If we assign $x(e) = \min\{1 - x[u], 1 - x[v]\}$ then we can ensure that $e$ satisfies the maximality condition along with ensuring that both $u$ and $v$ satisfy the feasibility condition. The only issue that can arise here is that some other edge adjacent to $u$ or $v$ is trying to update its value in parallel with edge $e$. Since we are looking at edge of type “End” and proceeding sequentially based on label $\ell \in L$, the above issue can only be caused by paths of length 2. However, the middle vertex of this path can decide the sequential order in which the two edges are considered, after which this issue is avoided.

Step 1 and 2 take a constant number of rounds. Step 3 takes $T(2d)$ rounds to run algorithm $\mathcal{A}$ on graph $G'$. Step 4 considers $O(\Delta^2)$ labels, and for an individual label $\ell$, it takes constant time to assign the values. Overall, the time taken for graph of maximum degree $\Delta = 2d + 1$ is given by the function $T(\Delta) \le c_1 + c_2(\Delta)^2 + T(\Delta - 1)$ for some constants $c_1$ and $c_2$. Since $T(\Delta - 1)$ is independent of $n$, $T(\Delta)$ is independent of $n$ as well. Thus, we have obtained a valid solution for the maximal fractional matching problem for graphs in $\mathcal{G}_{2d+1}$ using values from the set $S(d)$. □

**Lemma 4.3.** *Fix $d \in \mathbb{N}$. If $S(d)$ is sufficient to obtain the solution for $\mathcal{G}_{2d+1}$, then $S(d+1)$ is sufficient to obtain the solution for $\mathcal{G}_{2d+2}$.*

*Proof.* The proof for this theorem uses the same ideas as done in [2]. Consider any graph $G \in \mathcal{G}_{2d+2}$ and let $\mathcal{A}$ be the PN algorithm that uses values in $S(d)$ to compute a valid solution for graphs in $\mathcal{G}_{2d+1}$. We make use of the following definitions from [2]:

**Definition 4.4** (almost-saturating solutions)**.** A half-integral fractional matching $x\colon E \to \{0, \frac{1}{2}, 1\}$ is almost-saturating if the following conditions hold for each node $v$:

- If $x[v] = 0$, then $x[u] = 1$ for all neighbors $u$ of $v$.
- If $x[v] = 1/2$, then $x[u] = 1$ for at least one neighbor of $v$.

**Definition 4.5** (half-saturated edges)**.** Consider an almost-saturating solution $x\colon E \to \{0, \frac{1}{2}, 1\}$. An edge $e = \{u, v\}$ is:

- half-saturated if $x[u] = x[v] = 1/2$,
- fully-saturated if $x[u] = 1$ or $x[v] = 1$.

In [2] there is an algorithm that finds an almost-saturating solution in $O(\Delta^2)$ rounds. Let $\bar{x}$ denote the almost-saturating solution for $G$, and we let $G'$ to be the subgraph induced by the half-saturated edges; note that for each node $v$ there has to be at least one incident edge that is not half-saturated. Hence $G' \in \mathcal{G}_{2d+1}$, and we can apply $\mathcal{A}$ to produce a solution $x'$ for $G'$ using values in set $S(d)$. We can then extend domain of $x'$ to $E$ by setting $x'(e) = 0$ for $e \notin G'$. Setting $x(e) = \bar{x}(e) + x'(e)/2$ now gives a maximal fractional matching for the graph $G$. Moreover, $x(e) \in S(d+1)$. The number of rounds for graphs of degree $\Delta = 2d+2$ is given by $T(\Delta) \le c_1 + c_2(\Delta)^2 + T(\Delta - 1)$ for some constants $c_1$ and $c_2$. Since $T(\Delta - 1)$ is independent of $n$, $T(\Delta)$ is also independent of $n$. □

## 5 Conclusions

Our results give a complete characterization of how fine-grained fractional values are needed in a distributed algorithm that finds a maximal fractional matching in any running time $T(\Delta)$ that only depends on the maximum degree $\Delta$ and is independent of $n$. The main open question is if we can achieve this bound in time $T(\Delta) = O(\Delta)$, similar to [3], or if $O(\Delta)$-round algorithms need even more fine-grained fractional values.

## References

[1] Dana Angluin. Local and global properties in networks of processors. In *Proc. 12th Annual ACM Symposium on Theory of Computing (STOC 1980)*, pages 82–93. ACM Press, 1980. `doi:10.1145/800141.804655`.

[2] Matti Åstrand, Patrik Floréen, Valentin Polishchuk, Joel Rybicki, Jukka Suomela, and Jara Uitto. A local 2-approximation algorithm for the vertex cover problem. In *Proc. 23rd International Symposium on Distributed Computing (DISC 2009)*, volume 5805 of *Lecture Notes in Computer Science*, pages 191–205. Springer, 2009. `doi:10.1007/978-3-642-04355-0_21`.

[3] Matti Åstrand and Jukka Suomela. Fast distributed approximation algorithms for vertex cover and set cover in anonymous networks. In *Proc. 22nd ACM Symposium on Parallelism in Algorithms and Architectures (SPAA 2010)*, pages 294–302. ACM Press, 2010. `doi:10.1145/1810479.1810533`.

[4] Alkida Balliu, Sebastian Brandt, Juho Hirvonen, Dennis Olivetti, Mikaël Rabie, and Jukka Suomela. Lower bounds for maximal matchings and maximal independent sets. *Journal of the ACM*, 68(5), 2021. `doi:10.1145/3461458`.

[5] R Bar-Yehuda and S Even. A linear-time approximation algorithm for the weighted vertex cover problem. *Journal of Algorithms*, 2(2):198–203, 1981. `doi:10.1016/0196-6774(81)90020-1`.

[6] Leonid Barenboim, Michael Elkin, Seth Pettie, and Johannes Schneider. The locality of distributed symmetry breaking. In *2012 IEEE 53rd Annual Symposium on Foundations of Computer Science (FOCS 2012)*, pages 321–330, 2012. `doi:10.1109/FOCS.2012.60`.

[7] Leonid Barenboim, Michael Elkin, Seth Pettie, and Johannes Schneider. The locality of distributed symmetry breaking. *J. ACM*, 63(3), 2016. `doi:10.1145/2903137`.

[8] Manuela Fischer. Improved deterministic distributed matching via rounding. *Distributed Computing*, 33, 2020. `doi:10.1007/s00446-018-0344-4`.

[9] Mika Göös, Juho Hirvonen, and Jukka Suomela. Lower bounds for local approximation. *Journal of the ACM*, 60(5), 2013. `doi:10.1145/2528405`.

[10] Mika Göös, Juho Hirvonen, and Jukka Suomela. Linear-in-$\Delta$ lower bounds in the LOCAL model. *Distributed Computing*, 30(5):325–338, 2017. `doi:10.1007/s00446-015-0245-8`.

[11] Michal Hanckowiak, Michal Karonski, and Alessandro Panconesi. On the distributed complexity of computing maximal matchings. *SIAM Journal on Discrete Mathematics*, 15(1):41–57, 2001. `doi:10.1137/S0895480100373121`.

[12] Juho Hirvonen and Jukka Suomela. Distributed maximal matching: greedy is optimal. In *Proc. 31st Annual ACM SIGACT-SIGOPS Symposium on Principles of Distributed Computing (PODC 2012)*, pages 165–174. ACM Press, 2012. `doi:10.1145/2332432.2332464`.

[13] Amos Israeli and A. Itai. A fast and simple randomized parallel algorithm for maximal matching. *Information Processing Letters*, 22(2):77–80, 1986. doi:10.1016/0020-0190(86)90144-4.

[14] Nathan Linial. Locality in distributed graph algorithms. *SIAM Journal on Computing*, 21(1):193–201, 1992. doi:10.1137/0221015.

[15] Alain Mayer, Moni Naor, and Larry Stockmeyer. Local computations on static and dynamic graphs. In *Proc. 3rd Israel Symposium on the Theory of Computing and Systems (ISTCS 1995)*, pages 268–278. IEEE, 1995. doi:10.1109/ISTCS.1995.377023.

[16] Moni Naor. A lower bound on probabilistic algorithms for distributive ring coloring. *SIAM Journal on Discrete Mathematics*, 4(3):409–412, 1991. doi:10.1137/0404036.

[17] Alessandro Panconesi and Romeo Rizzi. Some simple distributed algorithms for sparse networks. *Distributed Computing*, 14, 2001. doi:10.1007/PL00008932.

[18] David Peleg. *Distributed Computing: A Locality-Sensitive Approach.* Society for Industrial and Applied Mathematics, 2000. doi:10.1137/1.9780898719772.

[19] A. Schrijver. *Combinatorial Optimization - Polyhedra and Efficiency.* Springer, 2003.

[20] Masafumi Yamashita and Tsunehiko Kameda. Computing on anonymous networks: part I—characterizing the solvable cases. *IEEE Transactions on Parallel and Distributed Systems*, 7(1):69–89, 1996. doi:10.1109/71.481599.