Tight Bounds on the Message Complexity of Distributed Tree Verification

Shay Kutten
Technion – Israel Institute of Technology, Haifa, Israel

Peter Robinson
Augusta University, GA, USA

Ming Ming Tan
Augusta University, GA, USA

Abstract

We consider the message complexity of verifying whether a given subgraph of the communication network forms a tree with specific properties both in the KT_ρ (nodes know their ρ-hop neighborhood, including node ids) and the KT_0 (nodes do not have this knowledge) models. We develop a rather general framework that helps in establishing tight lower bounds for various tree verification problems. We also consider two different verification requirements: namely that *every* node detects in the case the input is incorrect, as well as the requirement that *at least one* node detects. The results are stronger than previous ones in the sense that we assume that each node knows the number n of nodes in the graph (in some cases) or an α approximation of n (in other cases). For spanning tree verification, we show that the message complexity inherently depends on the quality of the given approximation of n: We show a tight lower bound of $\Omega(n^2)$ for the case $\alpha \geq \sqrt{2}$ and a much better upper bound (i.e., $O(n \log n)$) when nodes are given a tighter approximation. On the other hand, our framework also yields an $\Omega(n^2)$ lower bound on the message complexity of verifying a minimum spanning tree (MST), which reveals a polynomial separation between ST verification and MST verification. This result holds for randomized algorithms with perfect knowledge of the network size, and even when just one node detects illegal inputs, thus improving over the work of Kor, Korman, and Peleg (2013). For verifying a d-approximate BFS tree, we show that the same lower bound holds even if nodes know n exactly, however, the lower bounds is sensitive to d, which is the stretch parameter. First, under the KT_0 assumption, we show a tight message complexity lower bound of $\Omega(n^2)$ in the LOCAL model, when $d \leq \frac{n}{2+\Omega(1)}$. For the KT_ρ assumption, we obtain an upper bound on the message complexity of $O(n \log n)$ in the CONGEST model, when $d \geq \frac{n-1}{\max\{2,\rho+1\}}$, and use a novel charging argument to show that $\Omega\left(\frac{1}{\rho}\left(\frac{n}{\rho}\right)^{1+\frac{c}{\rho}}\right)$ messages are required even in the LOCAL model for comparison-based algorithms. For the well-studied special case of KT_1, we obtain a tight lower bound of $\Omega(n^2)$.

2012 ACM Subject Classification Theory of computation → Distributed algorithms

Keywords and phrases Distributed Graph Algorithm, Lower Bound

Digital Object Identifier 10.4230/LIPIcs.OPODIS.2023.26

Funding *Shay Kutten*: This research was supported in part by grant 2070442 from the ISF.

1 Introduction

Verifying the correctness of a given solution to a graph problem is an important problem with numerous applications. In this setting, there are n nodes that communicate via message passing over the edges of some arbitrary synchronous communication network G. Certain edges in G are labeled and the labeling of an edge e is part of the initial state of the nodes incident to e. For example, when considering the verification of a minimum spanning tree (MST), the label could indicate whether e is part of the MST, whereas for verifying a breadth-first search (BFS) tree, e's label may indicate the direction of the edge in the BFS tree T in addition to whether $e \in T$.

Generally, for a graph verification problem $\mathcal{P}$, we assume that the labels correspond to some (possibly weighted and/or directed) graph structure L of G. After observing the labels of their incident edges, the nodes may exchange messages with their neighbors and, eventually, every node needs to output 1 ("accept") if L corresponds to a *legal solution* to $\mathcal{P}$ for the communication network G. On the other hand, if L is *illegal*, we study two different requirements:

1. All-Detect: Every node outputs 0 ("reject").
2. One-Detects: At least one node outputs 0, whereas the other nodes may output either 0 or 1.

We point out that All-Detect and One-Detects have both been assumed in previous works. The One-Detects requirement, originally proposed in [1], is the basis of the area of distributed verification, e.g., see [19, 8, 25], whereas [17], which is closely related to the results of our work, considers All-Detect. We refer the reader to the survey of [7] for a more thorough survey of these results.

Apart from the spanning tree (ST) and minimum spanning tree (MST) verification problems, we also consider verifying approximate versions of breadth-first search (BFS) trees, which we define next: Consider a connected graph G and some subgraph $H \subseteq G$. We define $\mathsf{dist}_H(u, v)$ to denote the minimum hop distance of nodes u and v, when using only edges in H.

▶ **Definition 1.** *A spanning tree T is a* d-approximate Breadth-first Search Tree (d-approximate BFS) *of G if, for a designated root node u_0, the stretch of the shortest-path distance between u_0 and any other node in T is at most d. Formally, $\mathsf{dist}_T(u_0, v) \leq d \cdot \mathsf{dist}_G(u_0, v)$ for all nodes v in G.*

The input labeling for this problem is similar to that of BFS: it induces a directed subgraph and the labels should indicate, for each node, which one of its ports points to its parent and its children in T (if any).

Closely related to our work are the results of [17], which show that any deterministic distributed algorithm that verifies a minimum spanning tree without any knowledge of the network size and guarantees All-Detect must send $\Omega(m)$ messages in the worst case. We emphasize that the knowledge of n often strengthens algorithms and reduces their complexity, so our results that are given in spite of assuming nodes posses such knowledge (or, sometimes, the knowledge of an approximation of n) are stronger. The work of [30] proves that checking whether a given set of edges induces a spanning tree requires $\Omega(\sqrt{n} + \mathsf{Diam})$ rounds in the CONGEST model even for randomized algorithms and in fact, they prove that the same bound holds for a long list of fundamental graph verification and construction problems. Since [17] also gives a deterministic algorithm that has worst-case complexities of $\tilde{O}(\sqrt{n} + \mathsf{Diam})$ rounds, the time complexity of spanning tree verification is completely resolved, up to logarithmic factors. While the time complexity of distributed verification problems has been studied extensively in previous works, far less is known about the best possible bounds on the message complexity. To the best of our knowledge, the only result on the message complexity of verifying a tree problem was given in the aforementioned work of [17], where they prove that $\Theta(m)$ messages are essentially tight for MST verification. However, their lower bound technique only holds for deterministic algorithms that do not use any knowledge of the network size.

1.1 Our Contributions

We show almost-tight bounds on the message complexity of distributed verification for spanning trees and d-approximate breadth-first search trees. All our lower bounds for the clean network model (i.e. KT_0) hold even in the LOCAL model, whereas the upper bounds work in the CONGEST model.

- In Section 2, we present a general lemma for deriving message complexity lower bounds for graph verification problems under the KT_0 assumption, where nodes are unaware of their neighbors' IDs initially, which may be of independent interest.
- For **spanning tree (ST) verification**, we show that the knowledge of the network size n is crucial for obtaining message-efficient verification algorithms:
  - When nodes know the *exact* network size, we give a deterministic ST verification algorithm that guarantees the strong All-Detect property with a message complexity of only $O(n \log n)$ messages (see Theorem 19 on page 15).
  - If nodes have an α-approximation of n, for any $1 < \alpha < \sqrt{2}$, we still obtain $O(n \log n)$ messages, although we can only achieve One-Detects (i.e., at least one node detects illegal inputs). We prove that this is unavoidable by showing that All-Detect requires $\Omega(n^2)$ messages for any $\alpha > 1$ (see Theorem 10 on page 8).
  - On the other hand, we show that, when $\alpha \geq \sqrt{2}$, there is no hope for obtaining a message-efficient algorithm that guarantees One-Detects, as we prove that there are graphs with $\Theta(n^2)$ edges, where the message complexity is $\Omega(n^2)$.
- For **MST verification**, we show that $\Omega(n^2)$ poses an insurmountable barrier, as it holds for randomized algorithms, when nodes have perfect knowledge of the network size, and even under the weak requirement that just one node detects illegal inputs.
- For **verifying a d-approximate BFS tree**, we obtain the following results:
  - Under the KT_0 assumption, we prove that any randomized verification algorithm must send $\Omega(n^2)$ messages (Theorem 6 on page 6), for any $d \leq \frac{n}{2+\Omega(1)}$, even if the nodes have perfect knowledge of the network size. This bound is essentially tight in terms of the stretch d, as we also give an efficient algorithm in Section 5 that achieves $O(n \log n)$ messages, when $d \geq \frac{n}{2} - \frac{1}{2}$.
  - We also consider the d-approximate BFS verification problem under the KT_ρ assumption, for $\rho \geq 1$, where nodes are aware of their ρ-hop neighborhood initially, excluding the private random bits of the nodes. When d is small, we show that the lower bound of $\Omega(n^2)$ continues to hold in KT_1 for comparison-based algorithms. For $\rho \geq 2$ and any $d \leq O\left(\frac{n}{4\rho-2}\right)$, we develop a novel charging argument to show that $\Omega\left(\frac{1}{\rho}\left(\frac{n}{\rho}\right)^{1+\frac{c}{\rho}}\right)$ messages are required, which may turn out to be useful for proving lower bounds for other graph problems in KT_ρ, in particular, for $\rho \geq 2$ (see Theorem 12). We also show that the restriction on d cannot be improved substantially, by giving an upper bound of $O(n \log n)$ messages when $d \geq \frac{n-1}{\max\{2,\rho+1\}}$ (see Theorem 21 on page 15).

1.2 Additional Related Work

The research on ST, MST and BFS is too vast to provide a comprehensive survey. The d-approximate BFS tree problem is an important but limited (to a single source) version of the heavily studied spanner concept [29] of a subgraph with a few edges over which the distance of routing (here, just from the source) is an approximation of the original distance. The study of BFS approximation (in KT_0) has been motivated by the potential saving in the message and time complexities, especially when compared to those of the Bellman-Ford algorithm; see e.g., [2, 24, 6, 21, 12].

Tarjan [31, 14, 11, 5] considered the question of verifying a minimum weight spanning tree (MST) in the context of centralized computing, [16] addressed the problem in the context of PRAM, and [18] studied this question in the context of distributed computing with non-determinism, or with pre-processing. A verification algorithm may be a part of a fault-tolerant algorithm. Specifically, a verification algorithm can be executed repeatedly. If at some point, the verification fails, then an algorithm for re-computation is activated, followed again by repeated activations of the verification algorithm. In the context of self-stabilization, this was suggested in [13, 1] (the algorithms here are not self-stabilizing, though). More generally, in complexity theory and in cryptography, the issue of the complexity of verifying vs. that of computing is one of the main pillars of complexity theory, see, for example, the example of NP-hardness, Zero Knowledge, PCP, and IP (Interactive Proofs). In recent years there has been a lot of research in trying to adapt this kind of theory to distributed computing. We refer the reader to [19, 8, 25] for a more thorough survey of these results. It seems that the idea to verify a program while it is already running (as opposed to methods such as theorem proving, model checking, or even testing) appeared in general computing possibly after they were studied in distributed computing, but meanwhile, this has become a very developed area, see e.g. [22].

1.3 Preliminaries

We consider the standard synchronous CONGEST and LOCAL models [28], where all nodes are awake initially and communicate via message passing. Our main focus of this work is on the *message complexity* of distributed algorithm, which, for deterministic algorithms, is the worst-case number of messages sent in any execution. We assume that each node has a unique ID that is chosen from some polynomial range of integers.

When considering message complexity, the initial knowledge of the nodes becomes important: We follow the standard assumptions in the literature, which are KT_0, in the case where nodes do not know the IDs of their neighbors initially. Under the KT_0 assumption [3], which is also known as the *clean network model* [28], a node u that has δ neighbors also has bidirectional *ports* numbered $1, \ldots, \delta$ over which it can send messages; however, u does not know to which IDs its ports are connected to until it receives a message over this port. In contrast, the KT_1 assumption ensures that each node knows in advance the IDs of its neighbors and the corresponding port assignments. While it takes just a single round to extend KT_0 knowledge to KT_1, this would have required $\Omega(m)$ messages in general. Several algorithms have exploited the additional knowledge provided by KT_1 for designing message-efficient algorithms (e.g., [23, 15]).

2 A Framework for Message Complexity Lower Bounds in the KT_0 LOCAL Model

In this section, we present a general framework for deriving lower bounds on the message complexity of verification problems in the KT_0 LOCAL model.

We remark that the general framework is inspired by the bridge crossing lower bound of [20, 26, 27]. which, however, were designed for specific graph construction and election problems and do not apply to graph *verification*. We give some fairly general requirements for a hard graph and a corresponding labeling (see Definition 4) that, if satisfied, automatically yield nontrivial message complexity lower bounds. We start by introducing some technical machinery.

Rewireable Graphs, Rewirable Components, and Important Edges. Let H be a graph and $V(H)$ denotes that set of nodes in H. We use the notation $H[S]$ to denote the subgraph induced by a subset of nodes $S \subseteq V(H)$, and also define $L[S]$ to denote the labeling restricted to graph $H[S]$. We say that an n-node graph H is *rewirable* if there exist disjoint subsets $A_1, A_2 \subseteq V(H)$ such that $H[A_1]$ and $H[A_2]$ each contains at least an edge, but there are no edges between A_1 and A_2. We call $H[A_1]$ and $H[A_2]$ the *rewirable components* of H.

In our lower bounds, we identify two *important edges* $e_1 = (u_1, v_1) \in H[A_1]$ and $e_2 = (u_2, v_2) \in H[A_2]$. We define H^{e_1,e_2} to be the *rewired graph of* H on the same vertex set, by removing e_1 and e_2, and instead connecting A_1 and A_2 via these four vertices. Concretely, we have $E(H^{e_1,e_2}) = (E(H) \setminus \{e_1, e_2\}) \cup \{(u_1, v_2), (u_2, v_1)\}$, whereby (u_1, v_2) and (u_2, v_1) are connected using the same port numbers in H^{e_1,e_2} as for e_1 and e_2 in H.

▶ **Lemma 2.** *Let H be a rewirable graph. Then each node in H has an identical initial state in both H and H^{e_1,e_2} where H^{e_1,e_2} is any rewired graph of H.*

The proof is straightforward since rewiring does not change the degrees of the nodes and we are considering KT_0, where each node does not have information on the IDs of its neighbors.

We define $\mathsf{Inp}(G, L, \tilde{n})$ to denote the *input* where we execute the algorithm on graph G with the labeling L, and equip all nodes with the network size approximation $\tilde{n}$. An input $\mathsf{Inp}(G, L, \tilde{n})$ is said to be *legal* for problem $\mathcal{P}$ if L is a legal solution to $\mathcal{P}$ on the graph G. For a given algorithm $\mathcal{A}$, we say that inputs $\mathsf{Inp}(H, L, \tilde{n})$ and $\mathsf{Inp}(H', L', \tilde{n})$ are *indistinguishable for a node u* if u has the same probability distribution over its possible state transitions at the start of every round when $\mathcal{A}$ is executed on input $\mathsf{Inp}(H, L, \tilde{n})$ as it does on input $\mathsf{Inp}(H', L', \tilde{n})$. Formally, we write $\mathsf{Inp}(H, L, \tilde{n}) \cong \mathsf{Inp}(H', L', \tilde{n})$ if this indistinguishability is true for every node in the graph, and we use the notation $\mathsf{Inp}(H, L, \tilde{n}) \overset{S}{\cong} \mathsf{Inp}(H', L', \tilde{n})$ when this holds for every node in some set S. The following is immediate from the definition of indistinguishability and Lemma 2:

▶ **Lemma 3.** *Consider a graph H, a labeling L, and a rewired graph H^{e_1,e_2} of H. Let* Found *be the event that some node sends a message over an important edge, i.e., e_1 or e_2. Then, conditioned on event $\neg$*Found*, it holds that $\mathsf{Inp}(H, L, \tilde{n}) \cong \mathsf{Inp}(H^{e_1,e_2}, L, \tilde{n})$.*

▶ **Definition 4** (Hard Base Graph)**.** *Let $\tilde{n}$ be an α-approximation of the network size. We say that a rewirable graph H is a* hard base graph *for an algorithm $\mathcal{A}$ that solves a verification problem $\mathcal{P}$, if there is a labeling L and disjoint vertex sets S_1 and S_2, where $S_1 \cup S_2 = V(H)$ such that, for any important edges e_1 and e_2, the following properties hold:*

(A) $\mathsf{Inp}(H[S_1], L[S_1], \tilde{n}) \overset{S_1}{\cong} \mathsf{Inp}(H, L, \tilde{n}) \overset{S_2}{\cong} \mathsf{Inp}(H[S_2], L[S_2], \tilde{n})$.
(B) $\mathsf{Inp}(H[S_1], L[S_1], \tilde{n})$ *and* $\mathsf{Inp}(H[S_2], L[S_2], \tilde{n})$ *are legal for problem $\mathcal{P}$.*
(C) $\mathsf{Inp}(H^{e_1,e_2}, L, \tilde{n})$ *is illegal for $\mathcal{P}$.*

Remarks. Before stating our lower bound framework based on Definition 4, we provide some clarifying comments: Note that H does not need to be an admissible input to problem $\mathcal{P}$. In particular, H can be a disconnected graph even though problem $\mathcal{P}$ is defined for connected networks. We emphasize that S_1 and S_2 are not necessarily related to the rewirable components A_1 and A_2, introduced above. The difference is that A_1 and A_2 define where the important edges are selected from, whereas S_1 and S_2 partition $V(H)$ in a way such that both $\mathsf{Inp}(H[S_1], L[S_1], \tilde{n})$ and $\mathsf{Inp}(H[S_2], L[S_2], \tilde{n})$ are legal for problem $\mathcal{P}$ (even though $\mathsf{Inp}(H, L, \tilde{n})$ might not actually be an admissible input to problem $\mathcal{P}$).

Note that Property (A) is only relevant when the hard base graph H consists of multiple (i.e., disconnected) components. This will become apparent when proving a lower bound for d-approximate BFS tree verification in Section 2.1, where $S_1 = V(H)$ and $S_2 = \emptyset$, which makes the conditions on $H[S_2]$ stated in Property (A) and Property (B) vacuously true.

We prove the following result in Appendix A:

▶ **Lemma 5** (General KT_0 Lower Bound)**.** *Consider any ϵ-error randomized algorithm $\mathcal{A}$ for a graph verification problem $\mathcal{P}$, where $\epsilon < \frac{1}{6}$. If there exists a hard base graph H for problem $\mathcal{P}$ such that the rewirable components $H[A_1]$ and $H[A_2]$ are both cliques of size $\Theta(n)$, then $\mathcal{A}$ has an expected message complexity of $\Omega(n^2)$ in the KT_0 LOCAL model.*

2.1 A Lower Bound for d-Approximate BFS Verification

In this section, we assume that the labeling T is an arbitrary directed subgraph of the network G. The verification task requires checking whether T is indeed a d-approximate BFS tree of G (see Def. 1). Since the subgraph T is directed, each node in T knows its parents and children in T. We remark that this explicit specification of the direction of the tree (compared to the case where T is an undirected subtree of G) can only help the algorithm and hence strengthens our lower bound.

▶ **Theorem 6.** *Suppose that $d = \frac{n}{2} - \gamma$, for some $\gamma \leq \frac{n}{2} - 1$. Consider any ϵ-error randomized algorithm that solves d-approximate BFS tree verification in the KT_0 LOCAL model, and for any $\epsilon < \frac{1}{6}$. There exists an n-node network where the message complexity is $\Omega(\gamma^2)$ in expectation under the KT_0 assumption. In particular, for any $d \leq \frac{n}{2+\Omega(1)}$, this yields $\Omega(n^2)$ messages. This holds even when all nodes know the exact network size n.*

In the remainder of this section, we prove Theorem 6. To instantiate Lemma 5, we define a hard base graph H that satisfies Def. 4. For the sake of readability and since it does not change the asymptotic bounds, we assume that $2d$ and γ are integers. We group the vertices of H into $2d+2$ *levels* numbered $0, \ldots, 2d+1$. Levels 1 and $2d+1$ contain γ nodes each, denoted by $u_1^{(1)}, \ldots, u_\gamma^{(1)}$ and $u_1^{(2d+1)}, \ldots, u_\gamma^{(2d+1)}$, respectively. All the other levels consist of only a single node, and we use $u^{(i)}$ to denote the (single) node on level $i \in ([0, 2d] \setminus \{1\})$.

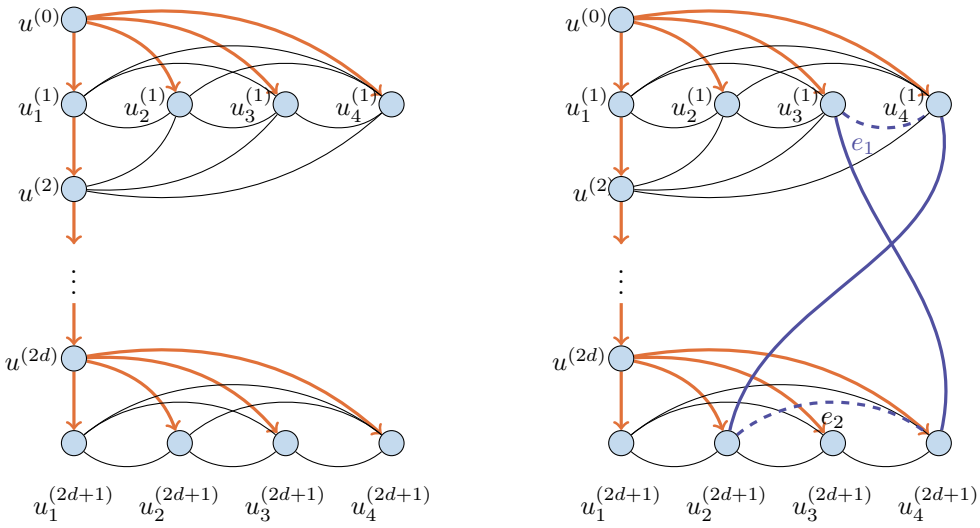
Next, we define the edges of H. Every node on level $i < 2d+2$ is connected via *inter-level* edges to every other node on level $i+1$. Moreover, the nodes on each level form a clique. Figure 1a gives an example of this construction.

To ensure that H is rewirable, we set A_1 to be the clique nodes on level 1 and A_2 to contain the clique nodes on level $2d+1$. We summarize the properties of H in the next lemma:

▶ **Lemma 7.** *Graph H has $2\gamma + 2d = n$ nodes and $\gamma^2 + 2\gamma + 2d - 2 = \Theta(\gamma^2)$ edges. It is a rewirable graph that contains two cliques, each on γ nodes, as rewirable components. The directed subgraph $T \subseteq H$ which contains all nodes of H and all directed inter-level edges is a d-approximate BFS tree of H.*

The next lemma tells us that the properties of Definition 4 hold, which will allow us to instantiate Lemma 5 for obtaining the sought lower bound.

▶ **Lemma 8.** *Consider the directed tree T defined in Lemma 7. Then, T is a d-approximate BFS tree of H, but not of any rewired graph H^{e_1,e_2} and, consequently, graph H is a hard base graph for d-approximate BFS verification with labeling T. This holds even if all nodes know the exact network size.*



(a) The hard base graph H for d-approximate BFS tree verification.

(b) The rewired graph H^{e_1,e_2} with the (dashed) important edges e_1 and e_2 that are replaced by the thick blue edges.

Figure 1 The graphs used in the lower bound for d-approximate BFS verification. The directed orange edges show the labeling, which is a BFS tree, and hence a legal input for any d.

Proof. We define $S_1 = V(H)$ and thus $S_2 = \emptyset$. For a given pair of important edges e_1 and e_2, we need to show Properties (A), (B), and (C) of Definition 4. Property (A) is trivial. For (B), note that $\mathsf{Inp}(H[S_1], L[S_1], n) = \mathsf{Inp}(H, T, n)$, and by Lemma 7, T is indeed a d-approximate BFS tree of H. For property (C), observe that the input tree T is not a d-approximate BFS tree of the rewired graph H^{e_1,e_2}, for any e_1 and e_2, because $\mathsf{dist}_T(u^{(0)}, u_k^{(2d+1)}) = 2d+1$, whereas $\mathsf{dist}_{H^{e_1,e_2}}(u^{(0)}, u_k^{(2d+1)}) = 2$ for $k \geq 1$. ◀

Theorem 6 follows by combining Lemma 7 and Lemma 8 with the general lower bound Lemma 5.

2.2 A Lower Bound for MST Verification

We now show that the lower bound graph used for d-approximate BFS tree verification in Section 2.1 is versatile enough to also yield a lower bound for MST verification.

▶ **Corollary 9.** *Suppose that all nodes know the exact network size. Let $\epsilon > 0$ be a sufficiently small constant. Any ϵ-error randomized algorithm that verifies whether an input is a minimum spanning tree in the* KT_0 LOCAL *model sends $\Omega(n^2)$ messages in expectation, even if the algorithm only ensures that at least one node detects illegal inputs (i.e.* $\mathsf{One\text{-}Detects}$*). The same result holds for verifying an approximate MST.*

Proof. We use a variant of the hard base graph H that we employed in the proof of Theorem 6, which is shown in Figure 1. The main difference is that now we consider an input labeling L that induces an undirected weighted subgraph, and we consider only 5 layers, where layer 1 and layer 4 are cliques, and the other layers consist of a single node each. Hence, there is a single edge $e = (u^{(2)}, u^{(3)})$ that acts as a bridge between the two cliques. The edge e has

weight $W > 1$ whereas every other edge in the graph has weight 0. The labeling L that we consider induces an MST that contains e and some arbitrary spanning tree on the rest of the graph. We now show that this satisfies Properties (A), (B), and (C) of Definition 4: We choose $S_1 = V(H)$ and $S_2 = \emptyset$, which makes Property (A) vacuously true. For Property (B), observe that $\mathsf{Inp}(H[S_1], L[S_1], n)$ corresponds to a valid MST of H and hence is legal. To see why Property (C) holds, it is sufficient to observe that any MST of the rewired graph H^{e_1,e_2} must include one of the rewired edges (connecting the two cliques) instead of the bridge edge e that has weight W. ◀

2.3 A Lower Bound for Spanning Tree Verification

We now consider the *Spanning Tree (ST) verification* problem where nodes have an α-approximation of the network size. The input is a connected graph G, a subgraph T of G, and an integer $\tilde{n}$ which is an α-approximation to the actual network size. The verification task requires checking distributively whether T is an ST of G, i.e., T is a tree that contains all nodes.

We remark that a message complexity lower bound of $\Omega(n^2)$ assuming All-Detect was previously shown by Kor, Korman, and Peleg [17] for deterministic algorithms in the setting where the network size is unknown. By using our framework, we generalize the message complexity lower bound to randomized algorithms and to the setting where nodes have an α-approximation of the network size. We also show that the bound depends on whether we assume One-Detects or All-Detect.

▶ **Theorem 10.** *Consider an ϵ-error randomized algorithm $\mathcal{A}$ that solves spanning tree verification in the KT_0 LOCAL model, where $\epsilon < \frac{1}{6}$, and suppose that nodes know an α-approximation of the network size:*

- *If all nodes detect an illegal input (i.e. All-Detect), then the message complexity is $\Omega(n^2)$ in expectation, for any $\alpha > 1$.*
- *If the algorithm satisfies only One-Detects, then the expected message complexity is still $\Omega(n^2)$, for any $\alpha \geq \sqrt{2}$.*

We emphasize that the bounds on the approximation ratio in Theorem 10 are tight, since in Section 4 we show that, if $\alpha < \sqrt{2}$, then the message complexity reduces drastically to just $O(n \log n)$ for One-Detects, and this holds even for All-Detect in the case where $\alpha = 1$.

Combining the following lemma with Lemma 5 immediately yields Theorem 10:

▶ **Lemma 11.** *There exists a graph H that is a hard base graph for ST verification with a suitable labeling where the rewireable components are cliques of size $\Omega(n)$, assuming the following restriction on the network size approximation known by the nodes:*

(i) *For* All-Detect*, this holds for any $\alpha > 1$.*
(ii) *For* One-Detects*, we require that $\alpha \geq \sqrt{2}$.*

Proof. We define a suitable lower bound graph H that consists of two (disconnected) cliques C and C'. We will specify the sizes of C and C' below. To obtain a labeling, we define T and T' be a spanning tree of the subgraphs C and C', respectively. To make H rewirable, we fix $A_1 = V(C)$ and $A_2 = V(C')$, i.e., the important edges will connect the cut (C, C') when being rewired.

We fix the sets $S_1 = V(C)$ and $S_2 = V(C')$ as required by Def. 4. For both cases, (i) and (ii), it is easy to see that Property (A) of Def. 4 holds since C and C' are (disconnected) components.

Thus, we only need to focus on Properties (B) and (C). First, we show Case (i) which is for All-Detect. We define C to be of size t and C' to be of size $(\alpha-1)t$,[1] which means that $n = \alpha t$. We equip nodes with the approximate network size $\tilde{n} = t$. It follows that both $H[A_1]$ and $H[A_2]$ contain $\Theta(n^2)$ edges. Consider the input $\mathsf{Inp}(H[S_1], L[S_1], \tilde{n}) = \mathsf{Inp}(C, T, t)$, which is legal since T is a spanning tree for C. This shows (B), since we consider All-Detect. For Property (C), consider any important edges e_1 and e_2. Graph H^{e_1,e_2} is connected, but the subgraph induced by the labeling $L := T \cup T'$ is not. Recalling that $V(H^{e_1,e_2})$ contains αt nodes, it follows that the network size approximation $\tilde{n} = t$ is indeed an α-approximation and thus the input $\mathsf{Inp}(H^{e_1,e_2}, T \cup T', t)$ is admissible for the algorithm and represents an illegal labeling. This completes the proof of (i).

Next, we show Case (ii) which is for One-Detects. Both C and C' contain exactly t nodes, and hence, again, it follows that H is rewirable. Here, we equip nodes with a network size approximation $\tilde{n} = \alpha t$. To show that Property (B) holds, we only need to prove that the network size approximation is admissible, since the rest of the argument is the same as for (i). Clearly $\mathsf{Inp}(H[S_1], L[S_1], \tilde{n}) = \mathsf{Inp}(C, T, \alpha\, t)$ and $\mathsf{Inp}(H[S_2], L[S_2], \tilde{n}) = \mathsf{Inp}(C', T', \alpha\, t)$ are both admissible and legal since C and C' contain t nodes each. Finally, For Property (C), fix any two important edges e_1 and e_2. We only need to argue that the input $\mathsf{Inp}(H^{e_1,e_2}, T \cup T', \alpha\, t)$ has an admissible network size approximation, as the rest of the argument that shows that it is not a valid spanning tree is the same as for (i). Observe that $n = |V(H^{e_1,e_2})| = 2t$ whereas $\tilde{n} = \alpha \cdot t$. It holds that $\tilde{n} \in [n/\alpha, \alpha\, n]$ if and only if

$$\frac{2t}{\alpha} \leq \tilde{n} = \alpha \cdot t \leq 2\alpha\, t.$$

In particular, the left inequality is true for any $\alpha \geq \sqrt{2}$, as required. ◀

3 A Lower Bound for d-Approximate BFS Verification in KT_ρ ($\rho \geq 1$)

Throughout this section, we consider comparison-based algorithms in the KT_ρ, CONGEST model. Apart from the result for broadcast of [3], we are not aware of any other superlinear (in n) message complexity lower bounds in the KT_ρ setting that hold for $\rho \geq 2$.[2] Even though it is relatively straightforward to generalize our approach to other verification problems such as spanning tree verification, here we exclusively focus on verifying a d-approximate BFS tree, which is more challenging in terms of techniques, as it requires us to exclusively deal with *connected* graphs. In particular, the charging argument in the lower bound approach of [3] crucially exploits the assumption that their "base" graph is disconnected.

In KT_ρ, a node x knows the IDs of all nodes that are at a distance less than or equal to ρ from x, and also knows the adjacencies of all nodes that have a distance of up to $\rho - 1$ from x. When considering the special case KT_1, a node simply knows the IDs of all its neighbors. Formally, we define $S_\rho(x) = \cup_{1 \leq r \leq \rho} C_r(x)$, where $C_r(x)$ consists of all nodes that are at distance r from x. Let $E_\rho(x) = \{(y,z) \in E | y, z \in S_\rho(x)\} \setminus \{(y,z) \in E | y, z \in C_\rho(x)\}$. The *$\rho$-neighborhood* of x is the subgraph $N_\rho(x) = (S_\rho(x), E_\rho(x))$.

We focus on comparison-based algorithms in the KT_ρ CONGEST model, which operate under the following restrictions: We assume that each node u has two types of variables. *ID variables* which contain u's neighborhood information, and *ordinary variables* that are

[1] For the sake of readability, we assume that $(\alpha-1)t$ is an integer. In the case that $(\alpha-1)t$ is not an integer, we set the size of G' to be $\lfloor(\alpha-1)t\rfloor$.

[2] We point out that [3] only provide a full proof of their result for the special case $\rho = 1$.

initialized to 0. During its local computation, a node may compare ID variables and it may perform some arbitrary computation on its ordinary variables. It can store the result of these computations in other ordinary variables. Each message sent by u may include $B = O(1)$ node IDs and it may also contain u's entire list of ordinary variables. Note that these are standard assumptions for comparison-based algorithms (see e.g., [3, 9, 27]). In the remainder of this section, we prove Theorem 12.

▶ **Theorem 12.** *Let $\beta > 0$ and $\epsilon > 0$ be suitable constants. Consider any $\rho \geq 1$, and $d \leq \frac{(1-\beta)n}{4\rho-2}$. For any ϵ-error randomized comparison-based algorithm that solves d-approximate BFS tree verification under the KT_ρ assumption with the guarantee that at least one node detects illegal inputs (i.e., $\mathsf{One\text{-}Detects}$), there exists a network where its message complexity is $\Omega\left(\frac{1}{\rho}\left(\frac{n}{\rho}\right)^{1+\frac{c}{\rho}}\right)$, for some constant $c > 0$. This holds even when all nodes know the exact network size. For the special case $\rho = 1$, we obtain a lower bound of $\Omega(n^2)$.*

The Lower Bound Graph Family

For a given $\rho \geq 1$ and $d \geq 1$, we construct an infinite family of n-node graphs (and their respective rewired variants) that yield the claimed bound of Theorem 12. We consider a graph H of n vertices that resembles the lower bound graph that sufficed for the KT_0 assumption (see Lemma 7). In the KT_ρ setting, a crucial difference is that, in order to show that the initial state of a node x has an order-equivalent ρ-neighborhood in H and in $H^{e,\tilde{e}}$, we need to add $\rho - 1$ layers before and after each of the rewirable components A_1 and A_2.

Figure 2 on page 12 gives an example of the graph construction that we now describe in detail: Let $\ell = (\rho + 1)d + \rho$. The vertices of H are partitioned into $\ell + 1$ *levels* numbered $0, \ldots, \ell$. Let $L = [1, 2\rho - 1]$ and $L' = [\ell - 2\rho + 2, \ell]$. The vertices of levels in L correspond to the nodes of the first $2\rho - 1$ layers after the layer 0. The vertices of levels in L' correspond to the nodes of the last $2\rho - 1$ layers. Let

$$k = \frac{1}{4\rho - 2} - \frac{1}{n} \cdot \frac{d(\rho + 1) - 3\rho + 3}{4\rho - 2} = \frac{1}{4\rho - 2} - O\left(\frac{d}{n}\right), \tag{1}$$

and let $\gamma = kn$.[3] For each $i \in L \cup L'$, level i contains γ nodes each, denoted by $u_1^{(i)}, \ldots, u_\gamma^{(i)}$. All the other levels consist of only a single node, and we use $u^{(i)}$ to denote the (single) node on level i for all $i \in [0, \ell] \setminus \{L \cup L'\}$.

Next, we define the edges of H: There is an edge from the root to every node in level 1. For each $i \in L \setminus \{1\} \cup L' \setminus \{\ell - 2\rho + 2\}$, there is an edge between nodes $u_j^{(i-1)}$ and $u_j^{(i)}$, for $1 \leq j \leq \gamma$. Let $M = [2\rho, \ell - 2\rho + 2]$. For each $i \in M$, all nodes at level i have an edge to every node at level $i - 1$; note that a level may only contain one such node. The nodes at level ρ form a subgraph C_ρ (defined below) and, similarly, the nodes at level $d(\rho + 1) + 1$ form another subgraph C'_ρ which is isomorphic to C_ρ. We call the edges that connect nodes at different levels the *inter-level edges*.

For a node $x \in C_\rho$ (resp. $x \in C'_\rho$), we use the notation $\tilde{x}$ to refer to its copy in C'_ρ (resp. in C_ρ), and we extend this notation to the edges in $C_\rho \cup C'_\rho$ in a natural way. An edge $e = (u, v) \in C_\rho$ induces the *rewired graph* $H^{e,\tilde{e}}$, where $\tilde{e} = (\tilde{u}, \tilde{v})$ is e's copy in C'_ρ. It is

[3] For the sake of readability, we assume that ℓ and γ are integers, which does not affect the asymptotic bounds.

straightforward to verify that each node in H has the same initial state in H and in $H^{e,\tilde{e}}$, since the ρ-neighborhood of each node x in C_ρ is isomorphic to the ρ-neighborhood $\tilde{x}$, by construction.

Let T be the directed tree, which contains all nodes of H and all directed inter-level edges. Then, T is indeed a (1-approximate) BFS tree of H. However, T is not a d-approximate BFS tree of $H^{e,\tilde{e}}$ because $\mathsf{dist}_T(u^{(0)}, u_k^{(d(\rho+1)+1)}) = d(\rho+1)+1$, whereas $\mathsf{dist}_{H^{e,\tilde{e}}}(u^{(0)}, u_k^{(d(\rho+1)+1)}) = \rho+1$ for $k \geq 1$. Hence the algorithm must produce different outputs on these two graphs when the input labeling is T.

The subgraph C_ρ

Intuitively speaking, the achieved message complexity lower bound will be determined by the number of edges in C_ρ, and thus we aim to make C_ρ as dense as possible. For $\rho = 1$, we simply define C_1 (and hence also C_1') to be the clique on γ vertices.

To simplify our argument needed for the lower bound result, we require that the subgraph C_ρ is such that if two nodes have distance at most ρ from each other, then there exists a unique path of that length between the two nodes, which we call the *ρ-unique shortest path* property. This trivially holds for $\rho = 1$ since C_1 and C_1' are cliques. For $\rho \geq 2$, we define C_ρ (and hence also C_ρ') to be a subgraph of γ nodes with girth greater than 2ρ and with $\Omega(\gamma^{1+\frac{c}{\rho}})$ edges for a constant $c > 0$. Recall that the *girth* is the length of the shortest cycle in a graph and that a graph with a girth greater than 2ρ has the ρ-unique shortest path property that we specified earlier. The existence of such graphs C_ρ is known due to [4].

We summarize the properties of H in the following lemma:

▶ **Lemma 13.** *The graph H contains subgraphs C_ρ and C_ρ' which satisfy the ρ-unique shortest path property, and $|V(C_\rho)| = |V(C_\rho')| = \gamma = kn$ such that k satisfies (1). If $\rho = 1$, the subgraph $C_1 \cup C_1'$ has $\Omega(n^2)$ edges. If $\rho \geq 2$, the subgraph $C_\rho \cup C_\rho'$ has $\Omega(\gamma^{1+\frac{c}{\rho}})$ edges for some constant $c > 0$. The directed tree T, which contains all nodes of H and all directed inter-level edges, is a d-approximate BFS tree of H but not of $H^{e,\tilde{e}}$.*

In our analysis, we make use of a particular notion of "connectedness" similar to the notion in [3]:

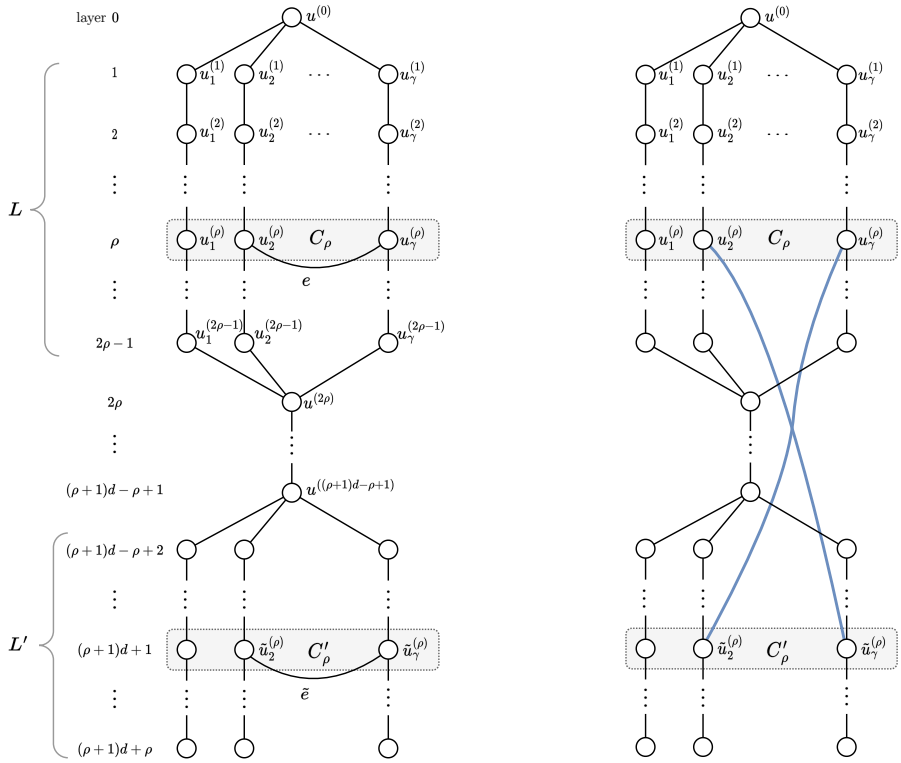
▶ **Definition 14** (e-connected)**.** *Consider an edge e in a graph that satisfies the ρ-unique shortest path property. We say that nodes x and z are e-*connected*, denoted by $x \overset{e}{-} z$, if $\mathsf{dist}(x,z) \leq \rho$ and the (unique) shortest path from x to z contains e.*

As outlined above, the proof of [3], for showing a lower bound on the message complexity of broadcast in KT_ρ, crucially relies on the fact that their base graph is disconnected. For showing a lower bound for the d-approximate BFS tree verification problem, we need the base graph as well as the rewired graphs to be connected. Thus, we need to introduce different rules for *utilizing* and *charging* edges, which have the added benefit of leading to a significantly simplified indistinguishability proof, as we will see in the proof of Lemma 18 below.

▶ **Definition 15** (utilized/unutilized edge)**.** *We say that an edge $e = (u,v) \in C_\rho \cup C_\rho'$ is* utilized *if one of the following conditions hold:*

1. *a message is sent across $e = (u,v)$;*
2. *there are two nodes x and z such that $x \overset{e}{-} z$, and x receives or sends the ID of z or $\tilde{z}$.*

Otherwise, we say that the edge e is unutilized*.*



(a) The base graph H.

(b) The rewired graph $H^{e,\tilde{e}}$.

Figure 2 The graph construction for proving a lower bound in KT_ρ for verifying a d-approximate BFS tree, where we have omitted the input labeling and node IDs. The shaded subgraphs are high girth graphs on γ nodes that have $\Omega\left(\gamma^{1+\frac{c}{\rho}}\right)$ edges. We emphasize that even the base graph H is connected, and thus the approach of [3] does not apply in our setting.

▶ **Definition 16** (charging rule). *If an edge (x, y) is used to send a message that contains the ID of node z, we charge one unit to each of the following edges:*

1. *(x, y);*
2. *for each edge e, such that either $x \overset{e}{-} z$ or $x \overset{e}{-} \tilde{z}$.*
3. *for each edge e, such that either $y \overset{e}{-} z$ or $y \overset{e}{-} \tilde{z}$.*

▶ **Lemma 17.** *Let μ be the number of utilized edges in an execution on H. Then the message complexity of the execution is $\Omega(\mu/\rho)$.*

Proof. Let μ be the number of utilized edges, M be the number of messages sent and C be the total cost charged in the execution. A message sent on an edge (x, y) will incur one unit of charge to (x, y) and for each z where its ID is in the message, a unit of charge is applied to each edge e where $x \overset{e}{-} z$, $x \overset{e}{-} \tilde{z}$, $y \overset{e}{-} z$ and $y \overset{e}{-} \tilde{z}$. Note that there are at most ρ edges of each of these types since the graph H satisfies the ρ-unique shortest path property. Note that each message can carry at most B IDs of other nodes. Consequently, for each of the messages sent, at most $1 + 4B\rho$ edges are charged. Hence, we have $C \leq (1 + 4B\rho)M$. On the other hand, each utilized edge is charged at least once. Hence, $\mu \leq C$. As a result, we have $M \geq \frac{\mu}{1+4B\rho}$ and the lower bound $\Omega(\mu/\rho)$ follows. ◀

Next, we present a suitable ID assignment ϕ for $V(H)$ $(= V(H^{e,\tilde{e}}))$ defined as follows:

1. assign a distinct even integer in $[0, 2(2\rho-1)\gamma]$ to each node in levels $1, \ldots, 2\rho-1$;
2. assign a distinct odd integer in $[1, 2(2\rho-1)\gamma+1]$ to each node x in levels $\ell-2\rho+2, \ldots, \ell$ such that $\phi(\tilde{x}) = \phi(x)+1$;
3. assign arbitrary unique integers from $[2(2\rho-1)\gamma+2, n]$ to the remaining nodes in $V(H)$.

We are now ready to present the indistinguishably result. For now, we focus on deterministic algorithms; we later extend the result to randomized algorithms via a simple application of Yao's lemma.

▶ **Lemma 18.** *Consider a deterministic comparison-based algorithm $\mathcal{A}$, the ID assignment ϕ, graph H and any rewired graph $H^{e,\tilde{e}}$, where $e \in C_\rho$. If e and $\tilde{e}$ are both unutilized (see Def. 15), then H and $H^{e,\tilde{e}}$ are indistinguishable for every node u when executing $\mathcal{A}$, i.e., u has an order equivalent initial state in both networks, and it sends and receives the same sequence of messages in H as it does in $H^{e,\tilde{e}}$.*

Proof. Let $e = (u, v)$. We first show that the statement holds for round 1, which is immediate for any node not in the $(\rho-1)$-neighborhood of either u, v, $\tilde{u}$, or $\tilde{v}$. Thus we focus on these nodes. We denote the set of these nodes as $N = N_{\rho-1}(u) \cup N_{\rho-1}(v) \cup N_{\rho-1}(\tilde{u}) \cup N_{\rho-1}(\tilde{v})$.

First, we show that the initial state of a node $x \in N$ has an order-equivalent $(\rho-1)$-neighborhood in H and in $H^{e,\tilde{e}}$. We observe that the $(\rho-1)$-neighborhood of x is isomorphic (ignoring the ID assignments) to the $(\rho-1)$-neighborhood of $\tilde{x}$. However, the IDs in the $(\rho-1)$-neighborhood of x in H are not the same as the IDs in the $(\rho-1)$-neighborhood of x in $H^{e,\tilde{e}}$. For instance, node u is adjacent to v in H but is adjacent to $\tilde{v}$ in $H^{e,\tilde{e}}$. Nevertheless, it follows from the definition of ϕ that the IDs in the $(\rho-1)$-neighborhood of x in H are order-equivalent to the IDs in the $(\rho-1)$-neighborhood of x in $H^{e,\tilde{e}}$, and thus x has order-equivalent neighborhoods in H and $H^{e,\tilde{e}}$.

Second, we show that each node sends the same messages in round 1. Since x has order-equivalent neighborhoods in both executions and recalling that the algorithm is comparison-based, any ordinary variable that x computes at the start of round 1 must have the same value in both executions. Next, we show that the ID type variable used for x in sending messages is identical in both executions as well. Recall that the IDs that x knows at the start of round 1 are exactly the IDs in its ρ-neighborhood, which may be different in H compared to $H^{e,\tilde{e}}$, as we have observed above. In particular, the IDs in $N_\rho(x)$ of H that are not in $N_\rho(x)$ of $H^{e,\tilde{e}}$ are the IDs of all vertices z, for which it holds that x and z are e-connected. The reason for this is that, in $H^{e,\tilde{e}}$, nodes x and $\tilde{z}$ are e-connected, whereas x and z are not. However, since e and $\tilde{e}$ are unutilized, x can neither include the ID of z in the execution on H, nor the ID of its neighbor $\tilde{z}$ in the execution on $H^{e,\tilde{e}}$, for any node z for which $x \overset{e}{-} z$. Hence, all IDs in the message sent by node x are identical in both executions. Therefore, x sends the same messages in both networks, H and $H^{e,\tilde{e}}$.

Finally, we show that if every node sends the same messages during round $r-1$, then every node sends the same messages in round r. We use induction over the rounds, where the basis already follows from above. Now consider some round $r > 1$ and assume that every node sent the same messages in round $r-1$ in both H and $H^{e,\tilde{e}}$. In this case, every node also receives the same messages during round $r-1$ in both executions. Consider the set of messages M received by x. Since e and $\tilde{e}$ are unutilized, no message in M contains the ID of z or $\tilde{z}$ for all nodes z that x is e-connected to. Hence, all IDs received by x correspond to the same nodes in H and $H^{e,\tilde{e}}$. Again, using the fact that the algorithm is comparison-based and the fact that all IDs that may be contained in the messages in M belong to nodes that have an identical ρ-neighborhood in H and $H^{e,\tilde{e}}$, it follows that x sends the same messages in round r. ◀

3.1 Proof of Theorem 12

First, consider any deterministic comparison-based algorithm. Assume towards a contradiction that there exists a deterministic comparison-based algorithm $\mathcal{A}$ that solves d-approximate BFS verification on input $\mathsf{Inp}(H, T, n)$ with message complexity $o\left(\frac{1}{\rho}\gamma^{1+\frac{c}{\rho}}\right)$. Then by Lemma 17, the number of utilized edges is $o(\gamma^{1+\frac{c}{\rho}})$. Lemma 13 tells us that there are $\Omega(\gamma^{1+\frac{c}{\rho}})$ edges in $C_\rho \cup C'_\rho$. Hence, there exists an edge e from C_ρ such that e and $\tilde{e}$ are unutilized. Consider the rewired graph $H^{e,\tilde{e}}$. Lemma 18 ensures that every node outputs the same result in both executions. However, according to Lemma 13, T is a d-approximate BFS of H but not of $H^{e,\tilde{e}}$, which provides a contradiction.

To obtain the claimed bound, we need to show that $\Omega\left(\frac{1}{\rho}\gamma^{1+\frac{c}{\rho}}\right) = \Omega\left(\frac{1}{\rho}\left(\frac{n}{\rho}\right)^{1+\frac{c}{\rho}}\right)$. By definition,

$$\begin{aligned}
\gamma &= kn \\
\text{(from (1))} \quad &= \frac{n}{4\rho-2} - \frac{d(\rho+1)}{4\rho-2} + \frac{3(\rho-1)}{2(\rho-2)} \\
\text{(since } \frac{\rho-1}{\rho-2} > 1\text{)} \quad &\geq \frac{n}{4\rho-2} - \frac{d(\rho+1)}{4\rho-2} \\
\text{(since } d \leq \frac{(1-\beta)n}{\rho+1}\text{)} \quad &\geq \frac{\beta n}{4\rho-2} = \Omega\left(\frac{n}{\rho}\right).
\end{aligned}$$

Randomized Algorithms. So far, we have restricted out attention to deterministic algorithms that succeed on every input. To extend this result to randomized Monte Carlo algorithms that fail with some small probability ϵ, we follow the standard approach of showing a lower bound for deterministic algorithms that succeed with a sufficiently large probability, when sampling the input graph from a hard distribution, defined as follows: We first flip a fair coin that determines whether we choose the base graph H or a rewired graph. In the latter case, we sample a rewired graph uniformly at random from the set of all possible rewired graphs $\mathcal{R}$. Observe that the algorithm cannot fail on graph H, since this would result in an error probability of at least $\frac{1}{2}$. Consequently, it is sufficient to show a lower bound under the assumption that the algorithm succeeds on a large fraction of the rewired graphs. We point out that a straightforward generalization of the above proof shows that the argument still holds for deterministic algorithms that succeed on a $(1-\beta)$-fraction of the graphs in $\mathcal{R}$ of the base graph H, for a sufficiently small $\beta > 0$. Thus, similarly to [27], a simple application of Yao's minimax lemma yields the sought message complexity lower bound for randomized Monte Carlo algorithms, and completes the proof of the theorem.

4 An Algorithm for Spanning Tree Verification in the KT_0 CONGEST Model

In this section, we give a message-efficient algorithm that verifies whether the input T is an ST of the network G under the One-Detects assumption, in the setting where all nodes have knowledge of some α-approximation $\tilde{n}$ of the network size n, for some $\alpha < \sqrt{2}$; formally speaking, $\tilde{n} \in [n/\alpha, \alpha n]$. Our result stands in contrast to the strong lower bound of $\Omega(n^2)$ shown by [17] that holds for deterministic ST verification assuming All-Detect and without *any* knowledge of the network size.

We obtain our algorithm by adapting the classic GHS algorithm for constructing an ST, see [10]. However, in contrast to the GHS algorithm, we do not employ the communication-costly operation of exchanging the fragment IDs between neighboring nodes, which requires

$\Omega(m)$ messages per iteration. Considering that our goal is to verify that a given tree T is indeed a spanning tree, we can select an arbitrary edge e from T (when growing a fragment) that is incident to some vertex of the fragment: we can stop the growing process immediately if e turns out to close a cycle. We prove the following result in Appendix B:

▶ **Theorem 19.** *Suppose that all nodes know an α-approximation of the network size, for some $\alpha < \sqrt{2}$. There is a deterministic KT_0 CONGEST algorithm that solves spanning tree verification with a message complexity of $O(n \log n)$ and a time complexity of $O(n \log n)$ rounds while ensuring at least one node detects illegal inputs (i.e., One-Detects). Moreover, if nodes have perfect knowledge of the network size, the algorithm guarantees All-Detect.*

While our main focus is on the message complexity, we point out that the round complexity of the algorithm in Theorem 19 can be as large as $O(n \log n)$. This comes as no surprise, considering that the state-of-the-art solution [23, 15] for computing a spanning tree with $O(n \operatorname{poly} \log n)$ messages (even under the stronger KT_1 assumption) requires at least $\Omega(n)$ rounds.

5 Algorithms for Verifying a d-Approximate BFS Tree

We now turn our attention to the d-BFS tree verification problem. The following lemma suggests that we can extend the algorithm for ST verification described in Section 4 by inspecting the neighborhood of the root when considering a sufficiently large stretch d:

▶ **Lemma 20.** *Let T be a d-approximate BFS tree of G with root r, and suppose that $d \geq \frac{n-1}{x+1}$, for some integer $x \geq 1$. If $\mathsf{dist}_T(r,u) > d \cdot \mathsf{dist}_G(r,u)$ for some node u in G, then $\mathsf{dist}_G(r,u) \leq x$.*

Proof. Consider a node u such that $\mathsf{dist}_T(r,u) > d \cdot \mathsf{dist}_G(r,u)$. It holds that

$$\mathsf{dist}_G(r,u) < \frac{\mathsf{dist}_T(r,u)}{d} \leq \frac{n-1}{d} \leq x+1,$$

and thus $\mathsf{dist}_G(r,u) \leq x$. ◀

▶ **Theorem 21.** *Consider the KT_ρ assumption, for any $\rho \geq 0$. If $d \geq \frac{n-1}{\max\{2,\rho+1\}}$, there exists a deterministic algorithm for d-approximate BFS verification that satisfies One-Detects with a message complexity of $O(n \log n)$ and a time complexity of $O(n \log n)$ rounds, assuming that nodes are given an α-approximation of the network size, for some $\alpha < \sqrt{2}$. If nodes have perfect knowledge of the network size, the algorithm ensures All-Detect.*

We point out that there is no hope of getting $O(n \log n)$ messages for significantly smaller values of d, as the lower bound in Theorem 12 holds for any $d \leq O\left(\frac{n}{4\rho-2}\right)$.

Proof of Theorem 21. First consider the case $\rho \in \{0,1\}$. Instantiating Lemma 20 with $x = 1$, tells us that we only need to check if T is a spanning tree and that all edges incident to the root are in T in order to verify if a subgraph T is a d-approximate BFS. More concretely, after executing the spanning tree verification, each node computes its distance in T from the root. Then the root directly contacts all its neighbors (in G): If there is a node at distance at

least $\frac{n-1}{2}$ from the root (in T) who is not a neighbor of the root, it outputs 0, and broadcasts a `fail` message to all nodes in T; otherwise, it broadcasts an `accept` message. Each node in T decides accordingly once it receives this message from its parent.

The argument for the case $\rho \geq 2$ is similar, except that we now instantiate Lemma 20 with $x = \rho$. That is, since the root knows its ρ-hop neighborhood, it can contact all nodes within distance ρ (in G) by using only $O(n)$ messages. ◀

6 Discussion and Open Problems

In this paper, we study the message complexity of ST verification, MST verification and d-approximate BFS verification distributed algorithms. To the best of our knowledge, the message complexity of d-approximate BFS verification distributed algorithms has never been studied before, hence, we focus our discussion on this problem. In our study, we show that the message complexity is largely determined by the stretch d. When d is small, we obtain a message complexity lower bound of $\Omega(n^2)$ for KT_0 and KT_1, and a lower bound of $\Omega\left(\frac{1}{\rho}\left(\frac{n}{\rho}\right)^{1+\frac{c}{\rho}}\right)$ (for some constant $c > 0$) for KT_ρ where $\rho > 1$. The bound of d is almost tight for KT_0 model, but not for KT_ρ. In particular, for KT_ρ model where $\rho > 1$, it is still open whether we can match the lower bound that holds for $d < \frac{n-1}{\max\{2,\rho+1\}}$.

In addition, all the bounds we obtain for KT_ρ where $\rho \geq 1$ are restricted to comparison-based algorithms, while the bound for KT_0 holds for general algorithms. This gives rise to the following important unanswered question: Can the lower bound of $\Omega(n^2)$ for KT_1 be improved by using non-comparison based algorithms?

References

1. Yehuda Afek, Shay Kutten, and Moti Yung. Memory-efficient self stabilizing protocols for general networks. In *Distributed Algorithms: 4th International Workshop Bari, Italy, September 24–26, 1990 Proceedings 4*, pages 15–28. Springer, 1991. doi:10.1007/3-540-54099-7_2.
2. Baruch Awerbuch, Amotz Bar-Noy, and Madan Gopal. Approximate distributed bellman-ford algorithms. *IEEE Transactions on Communications*, 42(8):2515–2517, 1994. doi:10.1109/26.310604.
3. Baruch Awerbuch, Oded Goldreich, Ronen Vainish, and David Peleg. A trade-off between information and communication in broadcast protocols. *Journal of the ACM (JACM)*, 37(2):238–256, 1990. doi:10.1145/77600.77618.
4. Béla Bollobás. *Extremal graph theory*. Courier Corporation, 2004.
5. Brandon Dixon, Monika Rauch, and Robert E Tarjan. Verification and sensitivity analysis of minimum spanning trees in linear time. *SIAM Journal on Computing*, 21(6):1184–1192, 1992. doi:10.1137/0221070.
6. Michael Elkin. Distributed approximation: a survey. *ACM SIGACT News*, 35(4):40–57, 2004. doi:10.1145/1054916.1054931.
7. Laurent Feuilloley, Pierre Fraigniaud, et al. Survey of distributed decision. *Bulletin of EATCS*, 1(119), 2016. URL: http://eatcs.org/beatcs/index.php/beatcs/article/view/411.
8. Pierre Fraigniaud, Amos Korman, and David Peleg. Towards a complexity theory for local distributed computing. *Journal of the ACM (JACM)*, 60(5):1–26, 2013. doi:10.1145/2499228.
9. Greg N Frederickson and Nancy A Lynch. Electing a leader in a synchronous ring. *Journal of the ACM (JACM)*, 34(1):98–115, 1987. doi:10.1145/7531.7919.
10. Robert G. Gallager, Pierre A. Humblet, and Philip M. Spira. A distributed algorithm for minimum-weight spanning trees. *ACM Transactions on Programming Languages and systems (TOPLAS)*, 5(1):66–77, 1983. doi:10.1145/357195.357200.

11 Dov Harel. A linear algorithm for finding dominators in flow graphs and related problems. In *Proceedings of the seventeenth annual ACM symposium on Theory of computing*, pages 185–194, 1985. doi:10.1145/22145.22166.

12 Monika Henzinger, Sebastian Krinninger, and Danupon Nanongkai. A deterministic almost-tight distributed algorithm for approximating single-source shortest paths. In *Proceedings of the forty-eighth annual ACM symposium on Theory of Computing*, pages 489–498, 2016. doi:10.1145/2897518.2897638.

13 Shmuel Katz and Kenneth Perry. Self-stabilizing extensions for message-passing systems. In *Proceedings of the ninth annual ACM symposium on Principles of distributed computing*, pages 91–101, 1990. doi:10.1145/93385.93405.

14 Valerie King. A simpler minimum spanning tree verification algorithm. *Algorithmica*, 18:263–270, 1997. doi:10.1007/BF02526037.

15 Valerie King, Shay Kutten, and Mikkel Thorup. Construction and impromptu repair of an MST in a distributed network with $o(m)$ communication. In *Proceedings of the 2015 ACM Symposium on Principles of Distributed Computing*, pages 71–80, 2015. doi:10.1145/2767386.2767405.

16 Valerie King, Chung Keung Poon, Vijaya Ramachandran, and Santanu Sinha. An optimal erew pram algorithm for minimum spanning tree verification. *Information Processing Letters*, 62(3):153–159, 1997. doi:10.1016/S0020-0190(97)00050-1.

17 Liah Kor, Amos Korman, and David Peleg. Tight bounds for distributed minimum-weight spanning tree verification. *Theory of Computing Systems*, 53(2):318–340, 2013. doi:10.1007/S00224-013-9479-7.

18 Amos Korman and Shay Kutten. Distributed verification of minimum spanning trees. In *Proceedings of the twenty-fifth annual ACM symposium on Principles of distributed computing*, pages 26–34, 2006. doi:10.1145/1146381.1146389.

19 Amos Korman, Shay Kutten, and David Peleg. Proof labeling schemes. In *Proceedings of the twenty-fourth annual ACM symposium on Principles of distributed computing*, pages 9–18, 2005. doi:10.1145/1073814.1073817.

20 Shay Kutten, Gopal Pandurangan, David Peleg, Peter Robinson, and Amitabh Trehan. On the complexity of universal leader election. *J. ACM*, 62(1):7:1–7:27, 2015. doi:10.1145/2699440.

21 Christoph Lenzen and Boaz Patt-Shamir. Fast routing table construction using small messages. In *Proceedings of the forty-fifth annual ACM symposium on Theory of computing*, pages 381–390, 2013. doi:10.1145/2488608.2488656.

22 Martin Leucker and Christian Schallhart. A brief account of runtime verification. *The journal of logic and algebraic programming*, 78(5):293–303, 2009. doi:10.1016/J.JLAP.2008.08.004.

23 Ali Mashreghi and Valerie King. Time-communication trade-offs for minimum spanning tree construction. In *Proceedings of the 18th International Conference on Distributed Computing and Networking, Hyderabad, India, January 5-7, 2017*, page 8. ACM, 2017. URL: http://dl.acm.org/citation.cfm?id=3007775.

24 Danupon Nanongkai. Distributed approximation algorithms for weighted shortest paths. In *Proceedings of the forty-sixth annual ACM symposium on Theory of computing*, pages 565–573, 2014. doi:10.1145/2591796.2591850.

25 Moni Naor, Merav Parter, and Eylon Yogev. The power of distributed verifiers in interactive proofs. In *Proceedings of the Fourteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1096–115. SIAM, 2020. doi:10.1137/1.9781611975994.67.

26 Shreyas Pai, Gopal Pandurangan, Sriram V. Pemmaraju, Talal Riaz, and Peter Robinson. Symmetry breaking in the congest model: Time- and message-efficient algorithms for ruling sets. In Andréa W. Richa, editor, *31st International Symposium on Distributed Computing, DISC 2017, October 16-20, 2017, Vienna, Austria*, volume 91 of *LIPIcs*, pages 38:1–38:16. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2017. doi:10.4230/LIPICS.DISC.2017.38.

27 Shreyas Pai, Gopal Pandurangan, Sriram V Pemmaraju, and Peter Robinson. Can we break symmetry with $o(m)$ communication? In *PODC'21: Proceedings of the 2021 ACM Symposium on Principles of Distributed Computing*, pages 247–257, 2021. doi:10.1145/3465084.3467909.

28 David Peleg. *Distributed computing: A locality-sensitive approach*. SIAM, 2000.

29 David Peleg and Alejandro A Schäffer. Graph spanners. *Journal of graph theory*, 13(1):99–116, 1989. doi:10.1002/JGT.3190130114.

30 Atish Das Sarma, Stephan Holzer, Liah Kor, Amos Korman, Danupon Nanongkai, Gopal Pandurangan, David Peleg, and Roger Wattenhofer. Distributed verification and hardness of distributed approximation. *SIAM Journal on Computing*, 41(5):1235–1265, 2012. doi:10.1137/11085178X.

31 Robert Endre Tarjan. Applications of path compression on balanced trees. *Journal of the ACM (JACM)*, 26(4):690–715, 1979. doi:10.1145/322154.322161.

A Proof of Lemma 5

▶ **Lemma 5** (restated). *Consider any ϵ-error randomized algorithm $\mathcal{A}$ for a graph verification problem $\mathcal{P}$, where $\epsilon < \frac{1}{6}$. If there exists a hard base graph H for problem $\mathcal{P}$ such that the rewirable components $H[A_1]$ and $H[A_2]$ are both cliques of size $\Theta(n)$, then $\mathcal{A}$ has an expected message complexity of $\Omega(n^2)$ in the* KT_0 LOCAL *model.*

Assume towards a contradiction, that there exists a randomized algorithm $\mathcal{A}$ that satisfies the premise of Lemma 5, while having an expected message complexity of $o(n^2)$. Consider a rewired graph H^{e_1,e_2} where the important edges e_1, e_2 are chosen uniformly at random from their respective rewirable components. Let Few be the event where at most $o(n^2)$ messages are sent and note that the bound on the expected message complexity ensures that Few happens with probability at least $1 - o(1)$. Recall that Found is the event that a message is sent over an important edge (see Lemma 3). We start our analysis by proving an upper bound on the probability that this happens.

▶ **Lemma 22.** $\mathbf{Pr}[\mathsf{Found}] \leq \frac{1}{2}$.

Proof. Since $\mathbf{Pr}[\neg\mathsf{Few}] = o(1)$, we have

$$\mathbf{Pr}[\neg\mathsf{Found}\,] \geq \mathbf{Pr}[\neg\mathsf{Found} \mid \mathsf{Few}]\,\mathbf{Pr}[\mathsf{Few}] \geq \mathbf{Pr}[\neg\mathsf{Found} \mid \mathsf{Few}] - o(1), \tag{2}$$

and hence it will be sufficient to show that $\mathbf{Pr}[\neg\mathsf{Found} \mid \mathsf{Few}] \geq \frac{1}{2} + \Omega(1)$.

We say that a port p incident to some node u is *unexplored*, if u has neither sent nor received a message over p. By assumption, the important edges $e_1 = (u_1, v_1)$ and $e_2 = (u_2, v_2)$ of the rewirable graph H are chosen uniformly at random such that $u_1, v_1 \in A_1$ and $u_2, v_2 \in A_2$. Let $n_i = |A_i|$, for $i \in \{1, 2\}$, and recall that $n_i = c_i\, n$, for some constant $c_i > 0$. Conditioned on Few, there must exist a subset $A_1' \subseteq A_1$ of size $(1 - o(1))n_1$, such that every $u \in A_1'$ sends and receives at most $\frac{n_1}{16}$ messages. Since A_1 is a clique and $n_1 = c_1 n$, every such u has at least

$$n_1 - 1 - \frac{n_1}{16} = \frac{15}{16} n_1 - 1 \geq \frac{7}{8} n_1$$

unexplored ports at any point in the execution, for sufficiently large n. A symmetric argument implies the existence of a set $A_2' \subseteq A_2$ with analogous properties. Let A^* be the event that $u_1, v_1 \in A_1'$ and $u_2, v_2 \in A_2'$. We first show that A^* happens with probability at least $1 - o(1)$. The probability that $u_1 \in A_1'$ is $\frac{|A_1'|}{|A_1|} \geq 1 - o(1)$, and analogous statements hold for v_1, u_2, and v_2. Hence, A^* occurs with probability $1 - o(1)$. Now, since $Pr[\neg A^*] = o(1)$, we have

$$\mathbf{Pr}[\neg\mathsf{Found} \mid \mathsf{Few}] \geq \mathbf{Pr}[\neg\mathsf{Found} \mid \mathsf{Few}, A^*] - o(1). \tag{3}$$

Conditioned on Few and A^*, we know that u_1 sends at most $\frac{n_1}{16}$ messages and has at least $\frac{7n_1}{8}$ unexplored ports, and thus the probability that it sends a message over the important edge e_1 is at most $\frac{1}{14}$. Moreover, u_2, v_1, and v_2 send a message over their incident important edge also with probability at most $\frac{1}{14}$. Thus, $\mathbf{Pr}[\neg\mathsf{Found} \mid \mathsf{Few}, A^*]$ is at least $1 - \frac{4}{14} = \frac{5}{7}$. Plugging this into the right-hand side of (3), we obtain

$$\mathbf{Pr}[\neg\mathsf{Found} \mid \mathsf{Few}] \geq \frac{5}{7} - o(1) = \frac{1}{2} + \Omega(1), \tag{4}$$

which is sufficient due to (2). ◀

For a given input $\mathsf{Inp}(G, L, \tilde{n})$ and a subgraph $X \subseteq G$, we define the event $Z[X]$ ("Zero output") as follows:

1. For $\mathsf{One\text{-}Detects}$, $Z[X]$ occurs if at least one node in X outputs 0.
2. For $\mathsf{All\text{-}Detect}$, $Z[X]$ occurs if all nodes in X output 0.

We slightly abuse notation and also write $Z[S]$ when considering the subgraph induced by a set of nodes $S \subseteq V(G)$.

According to Lemma 22, $\mathbf{Pr}(\mathsf{Found}) \leq \frac{1}{2}$, and thus

$$\begin{aligned}&\mathbf{Pr}(Z[H^{e_1,e_2}] \mid \mathsf{Inp}(H^{e_1,e_2}, L, \tilde{n})) \\ &\quad\leq \mathbf{Pr}(Z[H^{e_1,e_2}] \mid \mathsf{Inp}(H^{e_1,e_2}, L, \tilde{n}), \neg\mathsf{Found})\,\mathbf{Pr}(\neg\mathsf{Found}) + \tfrac{1}{2}.\end{aligned} \tag{5}$$

Property (C) of Definition 4 tells us that $\mathsf{Inp}(H^{e_1,e_2}, L, \tilde{n})$ is illegal, and hence the assumption that $\mathcal{A}$ fails with probability at most ϵ implies that

$$\mathbf{Pr}(Z[H^{e_1,e_2}] \mid \mathsf{Inp}(H^{e_1,e_2}, L, \tilde{n})) \geq 1 - \epsilon. \tag{6}$$

Therefore,

$$\begin{aligned}\mathbf{Pr}(Z[H^{e_1,e_2}] \mid \mathsf{Inp}(H^{e_1,e_2}, L, \tilde{n}), \neg\mathsf{Found}) &\geq \mathbf{Pr}(Z[H^{e_1,e_2}] \mid \mathsf{Inp}(H^{e_1,e_2}, L, \tilde{n}), \neg\mathsf{Found}) \\ &\qquad \cdot \mathbf{Pr}(\neg\mathsf{Found}) \\ \text{(by (5))} \quad &\geq \mathbf{Pr}(Z[H^{e_1,e_2}] \mid \mathsf{Inp}(H^{e_1,e_2}, L, \tilde{n})) - \tfrac{1}{2} \\ \text{(by (6))} \quad &\geq \tfrac{1}{2} - \epsilon.\end{aligned} \tag{7}$$

Conditioned on $\neg\mathsf{Found}$ (i.e., no important edge is discovered), Lemma 3 tells us that the algorithm behaves the same on the inputs $\mathsf{Inp}(H, L, \tilde{n})$ and $\mathsf{Inp}(H^{e_1,e_2}, L, \tilde{n})$, and hence (7) also yields

$$\mathbf{Pr}(Z[H] \mid \mathsf{Inp}(H, L, \tilde{n})) \geq \tfrac{1}{2} - \epsilon. \tag{8}$$

In Lemma 23 below, we show an upper bound of 2ϵ on the left-hand side of (8), which yields the sought contradiction $\epsilon \geq \frac{1}{6}$, and completes the proof of Lemma 5.

▶ **Lemma 23.** $\mathbf{Pr}(Z[H] \mid \mathsf{Inp}(H, L, \tilde{n})) \leq 2\epsilon$.

Proof. We start by observing that the following inequalities hold for the events $Z[S_1]$ and $Z[S_2]$:

$$\mathbf{Pr}[Z[S_1] \mid \mathsf{Inp}(H[S_1], L[S_1], \tilde{n})] \leq \epsilon. \tag{9}$$

$$\text{If } S_2 \neq \emptyset\text{: } \mathbf{Pr}[Z[S_2] \mid \mathsf{Inp}(H[S_2], L[S_2], \tilde{n})] \leq \epsilon. \tag{10}$$

These inequalities follow from Property (B) of Definition 4, which ensures $\mathsf{Inp}(H[S_1], L[S_1], \tilde{n})$ and $\mathsf{Inp}(H[S_2], L[S_2], \tilde{n})$ are both legal for $\mathcal{P}$. If $S_2 = \emptyset$, then

$$\mathsf{Inp}(H, L, \tilde{n}) = \mathsf{Inp}(H[S_1], L[S_1], \tilde{n}) \leq \epsilon$$

due to (9), and we are done. Thus, we assume that $S_2 \neq \emptyset$ in the remainder of the proof.

First, consider the case that the algorithm satisfies All-Detect, i.e., $Z[H]$ can only be true if $Z[S_1]$ and $Z[S_2]$ are both true: Using the indistinguishability guaranteed by Property (A) of Definition 4, the nodes in S_1 have the same probability distribution over their state transitions in $\mathsf{Inp}(H[S_1], L[S_1], \tilde{n})$ as they do in $\mathsf{Inp}(H, L, \tilde{n})$, and a similar argument applies to the nodes in S_2. From (9), (10), it follows that

$$\begin{aligned}\mathbf{Pr}[Z[H] \mid \mathsf{Inp}(H, L, \tilde{n})] &= \mathbf{Pr}[Z[S_1] \mid \mathsf{Inp}(H[S_1], L[S_1], \tilde{n})]\, \mathbf{Pr}[Z[S_2] | \mathsf{Inp}(H[S_2], L[S_2], \tilde{n})] \\ &\leq \epsilon^2 \leq 2\epsilon.\end{aligned}$$

Now, suppose that the algorithm satisfies One-Detects, where $Z[H]$ is true if either $Z[S_1]$ or $Z[S_2]$ are true. Again, using the indistinguishability guaranteed by Property (A), we obtain that

$$\begin{aligned}\mathbf{Pr}[Z[H] \mid \mathsf{Inp}(H, L, \tilde{n})] &\leq 1 - (1 - \mathbf{Pr}[Z[S_1] \mid \mathsf{Inp}(H[S_1], L[S_1], \tilde{n})]) \\ &\qquad \cdot (1 - \mathbf{Pr}[Z[S_2] \mid \mathsf{Inp}(H[S_2], L[S_2], \tilde{n})]) \\ \text{(by (9) and (10))} \quad &\leq 2\epsilon - \epsilon^2 \leq 2\epsilon.\end{aligned}$$

◀

B Omitted Details of Section 4

In the following, we describe and analyze the deterministic algorithm claimed in Theorem 19. We have omitted some proofs, which can be found in the full paper.

B.1 Description of the Algorithm

Growing Fragments

Each node forms the root of a directed tree, called *fragment*, that initially consists only of itself as the *fragment leader*, and every node in the fragment knows its (current) fragment ID, which is simply the ID of the fragment leader. The algorithm consists of iterations each comprising $c_1\,\tilde{n}$ rounds, for a suitable constant $c_1 \geq 1$, and the goal of an iteration is to merge the fragment with another fragment by finding an *unexplored edge*, i.e., some edge e over which no message has been sent so far. We point out that, in contrast to other ST construction algorithms that follow the Boruvka-style framework of growing fragments, e is *not* guaranteed to lead to another fragment.

In more detail, we proceed as follows: At the start of an iteration, each fragment leader broadcasts along the edges of its fragment F. Upon receiving this message from a parent in F, a node u checks whether it has any incident edges in T over which it has not yet sent a message. If yes, u immediately responds by sending its ID to its parent; otherwise, it forwards the request to all its children in F. If u does not have any children, it immediately sends a nil-response to its parent. On the other hand, if u does have children and it eventually receives a non-nil message from some child, it immediately forwards this response to its parent and ignores all other response messages that it may receive from its other children in this iteration. In the case where u instead received nil responses from every one of its children,

u finally sends a nil message to its own parent. This process ensures that the fragment leader u_ℓ eventually learns the ID of some node v in its fragment that has an unexplored incident tree edge $e_v \in T$, if such a v exists.

Next, u_ℓ sends an `explore!`-message along the tree edges that is forwarded to v, causing v to send a message including the fragment ID (i.e., u_ℓ's ID) on edge e_v, over which it has not yet sent a message. Assume that this edge is connected to some node w.

We distinguish two cases: First, assuming that w is in the same fragment as v, node w responds by sending an `illegal` message to v who upcasts this message to the fragment leader u_ℓ, who, in turn, initiates a downcast of this message to all nodes in the fragment, instructing every node in F to output 0 ("reject"). In this case, the fragment F stops growing and all its nodes terminate. Since we are assuming a synchronous network, it is clear that when nodes from other fragments happen to contact a node from F, they can detect this silence and will also immediately output 0 and terminate.

In the second case, w is in a distinct fragment, and, before we start the next iteration, we run a cycle detection procedure described next.

Acyclicity Check

Since all (non-terminated) fragments attempt to find outgoing edges in parallel in this iteration, we may arrive at the situation where there is a sequence of fragments $F_1, F_2, \ldots, F_k$ such that the unexplored edge found by F_i leads to F_{i+1}, for $i \in [1, k-1]$, and the unexplored edge discovered by F_k may lead "back" to some F_j $(j < k)$. Conceptually, we consider the *fragment graph* $\mathcal{F}$ where vertex f_i corresponds to the fragment leader of F_i, and there is a directed edge from f_i to f_j if the edge explored by F_i points to some node in F_j. (Note that, unlike the case in, e.g.[10], it is not guaranteed that no cycle is formed in the fragment graph, if T is not in fact, a tree.) If two fragments f and f' happen to both explore the same edge e in this iteration, then we say that e is a *core edge*.

▶ **Lemma 24.** *Every component C of $\mathcal{F}$ has at most one cycle. Moreover, C is a tree (i.e., contains no cycle) if and only there exist exactly two fragments f and f' in C that are connected by a core edge.*

Proof. If C contains just a single fragment, the statement is trivial; thus assume that there are at least two fragments in C. Since each fragment explores one outgoing edge, it is clear that C has at most one cycle, which proves the first statement. Now suppose that C is a directed tree. Recalling that each fragment in C has exactly one outgoing edge in $\mathcal{F}$, it follows that there must exist a core edge between two fragments, as otherwise there would be a cycle. Finally, since C has the same number of edges as it has fragments and is connected, it follows that there cannot be any other core edges. ◀

Lemma 24 suggests a simple way for checking whether C contains a cycle. We describe the following operations on $\mathcal{F}$. It is straightforward to translate these operations to the actual network G via broadcasting and convergecasting along the fragment edges. The fragment leaders first confirm with their neighboring fragments (in $\mathcal{F}$) whether they have an incident core edge. Every fragment leader that does not have an incident core edge simply waits by setting a timer of $t = c_2\tilde{n}$ rounds, for a suitable constant $c_2 > 0$. If there exists a core edge between f and f', then the leader with the greater ID, say f, broadcasts a `merge!` message, which is forwarded to all fragments in C by ignoring the direction of the inter-fragment edges, and is guaranteed to arrive within t rounds at every fragment leader of C. Upon receiving this message, every node in C adopts f's ID as its new fragment ID.

On the other hand, if the fragments form a cycle, then, after t rounds, all leaders in C conclude that there is no core edge in C. Lemma 24 ensures that there must be cycle. Thus, the nodes in C do not receive a `merge!` message, causing them to output 0, and terminate at the end of this iteration.

Check Size Requirement

Eventually, in some iteration, it may happen that the fragment leader u_ℓ receives nil messages from all its children, which means that none of the nodes in the fragment has any unexplored edges left. (Note that this also includes the special case where a node does not have any incident edges of T.) In that case, u_ℓ initiates counting the number of nodes in the fragment via a simple broadcast and convergecast mechanism. Once the counting process is complete, the root u_ℓ outputs 0 if the fragment contains less than $\frac{\tilde{n}}{\alpha}$ nodes, and it disseminates its output to all fragment nodes who in turn output 0 and terminate. Otherwise, u_ℓ instructs all nodes to output 1.