Near-Optimal Distributed Algorithms for Fault-Tolerant Tree Structures

Mohsen Ghaffari
MIT
ghaffari@mit.edu

Merav Parter
MIT
parter@mit.edu

ABSTRACT

Tree structures such as *breadth-first search* (BFS) trees and *minimum spanning trees* (MST) are among the most fundamental graph structures in distributed network algorithms. However, by definition, these structures are not robust against failures and even a single edge's removal can disrupt their functionality. A well-studied concept which attempts to circumvent this issue is *Fault-Tolerant Tree Structures*, where the tree gets augmented with additional edges from the network so that the functionality of the structure is maintained even when an edge fails. These structures, or other equivalent formulations, have been studied extensively from a centralized viewpoint [PP13, NSW97, CP10, GW12]. However, despite the fact that the main motivations come from distributed networks, their distributed construction has not been addressed before.

In this paper, we present distributed algorithms for constructing fault tolerant BFS and MST structures. The time complexity of our algorithms are nearly optimal in the following strong sense: they almost match even the lower bounds of constructing (basic) BFS and MST trees.

1. INTRODUCTION

Fault-Tolerant Tree Structures are well-studied concepts that attempt to add failure resilience to standard spanning tree structures. Spanning trees are among the most basic infrastructures used for communication in distributed networks. Two key examples are BFS and MST: Breadth First Search (BFS) trees are the main backbone in many communication tasks such as broadcast and routing as they provide the shallowest possible tree and shortest source-wise paths; Minimum Spanning Trees (MST) gets used because they provide the cheapest structure that keeps the graph connected. A downside of spanning trees is their sensitivity to faults, because even the failure of a single link disconnects the tree, which may lead to a complete loss of functionality. As the vertices and edges of the network may occasionally fail or malfunction, it is desirable to make these tree structures robust to failures.

A *Fault-Tolerant (FT) Tree Structure* augments the tree with additional edges from the base graph G in a way that provides it some redundancy and flexibility. As this augmentation carries certain costs, it is desirable to minimize the number of added edges. The motivation for this arises in settings where the graph edges represent the channels of a communication network, and the system designer would like to purchase or lease a minimal collection of channels (i.e., a subgraph $G' \subseteq G$) that maintain its functionality under faults. Another motivation for pre-computing fault tolerant tree structures arises in scenarios of *transient edge failures*: here, re-computing the structure after each failure is inefficient, as the failures are temporary, and instead one has to be prepared for each single edge's removal.

Indeed, time-efficient constructions of sparse FT tree structures have been studied extensively in the *centralized* setting for various notions of trees, see e.g., [NSW97, GW12, PP13, BK13, PP14, BGLP14]. Yet, albeit the fact that the key motivations come from distributed networks and these structures can be considered as *reliable* skeletons for distributed communication, their distributed construction has not been studied thoroughly before. In this paper, our objective is to provide *fast* distributed constructions of *sparse* FT tree structures. We next formalize this:

Model: We work with the standard distributed message passing model called CONGEST [Pel00]: communications occur in synchronous rounds; per round $O(\log n)$ bits can be sent along each link.

Problem Definitions: We focus on two fundamental tree structures, BFS and MST. The fault tolerant analogues of these structures are defined as follows: A *fault-tolerant BFS* (FT-BFS) structure is a sparse subgraph H of the given network G and a source node s that contains a BFS rooted at s in $G \setminus \{e\}$ for every failing of the edge $e \in G$. Thus, subsequent to the failure of a single edge $e \in G$, the surviving part $H \setminus \{e\}$ of H still contains a BFS tree of $G \setminus \{e\}$. Moreover, H preserves the source-wise distances as in the original graph G following the failing of a single edge. Similarly, a *fault-tolerant MST* (FT-MST) structure is a sparse subgraph H of the given network G that contains an MST in $G \setminus \{e\}$ for every failing of the edge $e \in G$. Given these definitions, our problems boil down to the following:

The research in this work was partially funded by the following grants: AFOSR FA9550-13-1-0042, NSF CCF-1461559, CCF-1217506, CCF-0939370.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

SPAA '16, July 11-13, 2016, Pacific Grove, CA, USA
© 2016 ACM. ISBN 978-1-4503-4210-0/16/07...$15.00
DOI: http://dx.doi.org/10.1145/2935764.2935795

***Core Problem**: Distributedly construct BFS trees (or MST trees) of graphs $G \setminus \{e\}$ for all $e \in G$.*

Although this specification suggests we need to compute m trees, where m is the number of edges of G, in reality we just need to compute n (potentially) distinct trees: one is the BFS or MST tree T_0 of the base graph G, and $n-1$ additional BFS or MST trees, one for $G \setminus \{e\}$ for each $e \in T$.

We use the standard distributed representation for trees. In particular, for BFS trees, each vertex knows its parent. Our algorithms also provide the additional information of each vertex knowing its children in the tree, and its distance from the source. For MST trees, in the output, every vertex knows which of its incident edges are in the MST.

Our Contribution: We show somewhat surprisingly that these sparse fault tolerant tree structures can be computed within almost the same number of rounds as the corresponding *non-resilient* counterparts, i.e., the time to compute just one of the trees. We present distributed algorithms for constructing FT-BFS in $O(D \log n)$ rounds and FT-MST $O(D + \sqrt{n} \log n)$. These round complexities are nearly optimal in the sense they almost match even the lower bounds of constructing BFS and MST trees, respectively. Our constructions lead to sparse structures by simple connections with previous works. Hence, our main focus would be on the running time.

Theorem 1.1 (**Distributed FT-BFS**). *Given an unweighted undirected n-vertex graph $G = (V, E)$ of diameter D, the collection of m BFS structures $\texttt{BFS}(G \setminus \{e\}, s)$ for every $e \in G$, containing at most $O(n^{3/2})$ edges, can be constructed in $O(D \log n)$ rounds, with high probability*[1].

Theorem 1.2 (**Distributed FT-MST**). *Given a weighted undirected n-vertex graph $G = (V, E, W)$ of diameter D, the collection of m MST trees $\texttt{MST}(G \setminus \{e\})$ for each $e \in G$, containing at most $2(n-1)$ edges, can be constructed in $O(D + \sqrt{n} \log n)$ rounds, with high probability.*

The above two round complexity bounds are nearly optimal, because of the following: computing even one BFS requires at least $\Omega(D)$ rounds (a trivial and folklore lower bound), and computing even one MST requires at least $\Omega(D + \sqrt{\frac{n}{\log n}})$ rounds, by lower bounds of Das Sarma et al. [DSHK+11] and its predecessors [PR99, Elk04].

We also note that our $O(D \log n)$ round FT-BFS construction leads to $O(D \log n)$ round distributed algorithms for single-source replacement paths and also BFS sensitivity analysis, and our $O(D + \sqrt{n} \log n)$ round FT-MST construction leads to an $(D + \sqrt{n} \log n)$ round distributed algorithm for MST sensitivity analysis. The definition of these problems are discussed in the next section.

2. RELATED WORK

2.1 Works Related to FT-BFS Structures

The notion of sparse FT-BFS structure was introduced in [PP13]. The authors showed that one can obtain an FT-BFS structure with $O(n^{3/2})$ edges, simply by doing the following; take the union of m BFS structures, one for subgraph $G \setminus \{e\}$, for each $e \in E$, while breaking ties in a consistent manner. They also provide a matching lower bound, by presenting a family of graphs for which $\Omega(n^{3/2})$ edges are necessary.

The FT-BFS structure is closely related to the problem of constructing *replacement paths*. For a pair of vertices s and t and a an edge e, the replacement path $P_{s,t,e}$ is the $s-t$ shortest path that avoids e. In the *replacement-path* problem, one is given a graph $G = (V, E)$ and a pair of vertices $s-t$, it is then required to compute the collection of all replacement paths $P_{s,t,e}$ for every edge e. s

For undirected graphs on m edges, Malik et al. [MMG89] gave an algorithm for solving the replacement path problem with time complexity of $\widetilde{O}(m)$. Gotthilf and Lewenstein [GL09] showed an $O(mn \cdot n^2 \log \log n)$ time algorithm for directed graphs with arbitrary edge weights. For unweighted directed graphs, Roditty and Zwick [RZ12] gave a randomized algorithm which runs in time $\widetilde{O}(m\sqrt{n})$. Weimann and Yuster [WY13] were the first to apply fast matrix multiplication techniques to the problem and obtained an $O(Mn^{2.584})$-time randomized algorithm for directed graphs with integer weights in $[-M, M]$. In [Wil11], Vassilevska Williams showed that the replacement paths problem in directed graphs is equivalent to the all pairs shortest paths problem (APSP), under subcubic reductions. She also provided a deterministic algorithm for this problem with improved rum time of $\widetilde{O}(Mn^{\omega})$ where ω is the exponent of matrix multiplication.

FT-BFS structures are in particular related to the *single source* variant of the replacement-path problem, namely, the *single-source replacement paths* problem, studied in [GW12]. That problem requires to compute the collection $\mathcal{P}_s$ of all $s-t$ replacement paths $P_{s,t,e}$ for every $t \in V$ and every failed edge e that appears on the $s-t$ shortest-path in G. Grandoni and Vassilevska Williams [GW12] showed that the single source replacement path problem can be solved within nearly the same centralized time complexity as that of computing all-pairs shortest paths in a graph. Since the collection of paths $\mathcal{P}_s$ is precisely an FT-BFS structure with respect to s, the best algorithm for computing an FT-BFS structure in the centralized setting is almost as that of computing the all pairs distances.

The replacement-path problem and its single source variant have been studied so far only in the centralized setting. These problems have applications for computing the k shortest simple $s-t$ paths [RZ12] and for computing the *Vickrey pricing* of edges in a network [NR99, HS01]. We note that our distributed construction of a sparse FT-BFS structure can also be viewed as a distributed computation of the single-source replacement paths.

2.2 Works related to FT-MST Structures

FT-MST structures were introduced by Nardelli et al. [NSW97] for the geometric setting. They showed that the problem of computing the minimum (with respect to total edge weights) FT-MST structure is NP-hard already for the single failure case. This problem has been later studied by Chechik and Peleg [CP10] in the *general* graph setting where they showed an $O(\log n)$-approximation algorithm for an arbitrary constant number f of edge faults.

The first distributed construction of FT-MST structures resilient against single *vertex* or edge faults was given by Flocchini at el. [FEP+12]. They provide an $O(n)$-round al-

[1] As standard, we use the phrase *with high probability* (w.h.p.) to indicate that the probability of an event is at least $1 - 1/n^c$, for a desired fixed constant $c \geq 2$.

gorithm with message complexity of $O(n^2)$. Linear time algorithm for constructing FT-MST (via the edge swapping approach) was also given by Datta et al. [DLPP13]. Moreover, Gfeller et al. [GSW11] provide $O(n)$-round algorithm for finding the fault tolerant MST of minimum diameter.

Finally, the concept of FT-BFS and FT-MST structures is closely related to the *sensitivity analysis problem* [Tar82]. In this problem, one is given an optimal tree (with respect to some function), and it is required to measure by how much one can perturb each edge individually (e.g., by changing the weight of the edge or deleting it entirely) without changing the optimality of the tree (compared to other tree solutions in the graph). The best centralized bounds for computing an FT-MST structure are given by solving the MST sensitivity problem: given a graph G and minimum spanning tree $T = MST(G)$, decide how much each individual edge weight can be perturbed without invalidating the identity of T_0 (i.e., that T_0 is still an MST for the perturbed graph). Tarjan showed that the MST sensitivity analysis can be perform in $O(m\alpha(m,n))$ time, where m in the number of edges, n is the number of vertices, and α is the inverse-Ackermann function. This was later improved by Pettie to an $O(m\log(\alpha(m,n)))$ time algorithm [Pet05]. Our distributed constructions for FT-BFS and FT-BFS can also be viewed as solving the sensitivity analysis problems for BFS and MST trees.

3. FAULT-TOLERANT BFS

In this section, we explain a distributed algorithm that given a source node s, in $O(D\log n)$ rounds computes an FT-BFS structure rooted in s that is tolerant against the failure of each one edge. Recall that to compute an FT-BFS structure, we need to compute BFS trees in graph $G\setminus\{e\}$ for each edge $e \in G$, which as we discuss next, boils down to computing only n BFSs. By [PP13], it is sufficient to break shortest-paths ties consistently. To do that, in our distributed BFS construction, each node would break the ties among its potential parents in the BFS tree by selecting the one of minimum ID.

A basic BFS tree $T_0 = BFS(G,s)$ rooted in node s can be computed in $O(D)$ rounds, using the standard approach of synchronously growing the BFS-tree one hop per round, starting from the source s. Besides this base BFS tree, what remains to be computed is $n-1$ additional BFS trees rooted in s, one in each graph $G\setminus\{e\}$ for each edge $e \in T_0$.

As a side remark, we note that this problem is reminiscent of the problem of computing all-to-all shortest paths in G, i.e., computing n BFS trees, one rooted in each node $v \in G$. Holzer and Wattenhofer [HW12] showed how to compute these n BFS trees simultaneously in $O(n)$ rounds, using a simple and elegant scheduling idea. We will show that in our case, the $n-1$ BFSs can be computed much faster in $O(D\log n)$ rounds, thanks to their structure.

3.1 Naive Approaches

We first discuss two simpler and natural approaches that do not leverage the special structure of these BFSs and lead to results much weaker than our end goal of $O(D\log n)$.

Straw Man Approach 1: The most naive method for solving this problem would be to run the BFSs one by one, sequentially. That would lead to a round complexity of $O(Dn)$, a far cry from our goal of $O(D\log n)$ rounds.

Straw Man Approach 2: The second method would be to use the classic idea of *random delays* due to Leighton, Maggs, and Rao [LMR94,Gha15] to schedule the $n-1$ BFSs. Particularly, we would divide time into phases of $O(\log n)$ rounds, and delay the start of each BFS T_e by a random delay of t_e phases where t_e is a random variable with a uniform distribution in $\{1, 2, \ldots, \frac{n}{\log n}\}$. Once a BFS algorithm starts, the related BFS grows at a synchronous speed of one hop per phase. One can see that in this way, w.h.p., per phase and per edge $e' = (v,u)$, there are at most $O(\log n)$ BFS tokens scheduled to go through edge e' from v to u, in this phase. The reason is as follows: for each BFS, the phase number in which the BFS arrives at v and goes to u is exactly equal to the distance of v from s in the corresponding graph plus the random delay of that BFS. Now due to the fact that the random delay is picked uniformly from a range of size $\frac{n}{\log n}$, the probability of this summation being equal to each exact given value is at most $\log n/n$. Hence, over all the n BFSs, we expect to have at most $\log n$ BFS token that try to go from v to u per phase. By Chernoff bound, we know the number of such BFS tokens is at most $O(\log n)$, with high probability. Since each phase has $O(\log n)$ rounds, this means the phase has enough capacity to admit all the messages of all the BFSs through e. Thus, once started, the BFSs will be able to continually grow with the speed of one hop per phase. Therefore, the finishing time of this routine is $O((D + \frac{n}{\log n}))$ phases, which is $O(D\log n + n)$ rounds.

This bound is again far from our goal of $O(D\log n)$ rounds, due to the additive $O(n)$ term. This undesirable additive $O(n)$ term comes from the range of the random delay, which was $\frac{n}{\log n}$ phases. This range size was chosen to ensure that per phase no more than $O(\log n)$ of the n BFSs try to go through each edge. In fact, this is unavoidable unless we change the BFS algorithms, because if for an edge e', each of the $n-1$ BFSs should send one messages through e', any scheduling of these BFSs clearly requires $n-1$ rounds.

As a side note, we remark that albeit its shortcoming in our case, the above simple and general approach is quite useful. Particularly, it can be used to compute all-to-all shortest paths in $O(D\log n + n)$ rounds, which is only slightly worse than the $O(n)$ result of [HW12].

3.2 The Intuition of Our Approach

To get to the bound of $O(D\log n)$, we will have to zoom in on the BFSs that we are trying to build and leverage their structure. Particularly, as noted above, it is vital to make sure that per edge e, not all the $n-1$ BFSs need to transmit independent messages through edge e—as that would require at least $n-1$ rounds. For that, consider one edge $e' = (v,u)$, and let us zoom in on what messages need to be sent from v to u for various BFSs.

In a synchronous growth of a BFS tree $T_e = BFS(G\setminus e, s)$, the key thing that matters for each node v is when is the first time that the BFS token arrives at v (as well as who is the related parent). Note that this time indicates the distance of v from the source, in the corresponding graph. For edge $e' = (v,u)$, we need to send the BFS token from v to u if in $G\setminus\{e\}$, node v is closer to s compared to u. Moreover, in that case, what u needs to learn is the time that this token will be sent from v to u. This time is equal to $dist_{G\setminus\{e\}}(s,u) + t_e$, where the latter term is the starting time of the BFS T_e. A key observation is that, for many of the $n-1$ BFSs T_e that we are trying to run, we can make u know this time $dist_{G\setminus\{e\}}(s,u) + t_e$ without actually running that BFS through edge e. Note that in each tree T_e, node u receives messages from all its potential parents at the same time, namely, at time $dist_{G\setminus\{e\}}(s,u)+t_e$. It would then select as its parent the node of minimum ID, which would guarantee that the shortest path ties are broken an a consistent manner, leading to a sparse structure with $O(n^{3/2})$ edges.

Let P_v be the shortest path connecting s to v in G. If $e \notin P_v$, then $dist_{G\setminus\{e\}}(s,v) = dist_G(s,v)$. So, for each $e \notin P_v$, the $BFS(G\setminus\{e\},s)$ arrives at v at time $dist_G(s,v)+t_e$. Thus, we can avoid running all these BFSs through edge e if we could make node u know three things: (1) the distance $dist_G(s,v)$ in the base graph G, (2) the edges e on shortest path P_v in the base graph G, (3) the starting time for $BFS(G\setminus\{e\},s)$ for each of these $e \notin P_v$.

We will explain that we will be able to make each node u have this information about each of its neighbors v, using just $O(D+\log n)$ rounds of communication. Hence, the only BFSs that need to actually send a token through edge $e'=(v,u)$ from v to u are those T_e for $e \in P_v$. This is a total of D many BFSs, so using the random delays technique, we will be able to schedule these remaining BFSs in a short span of time. However, having these random delays creates a complication, as we need to make nodes know the starting time of (essentially) all BFSs, as needed in item (3) above. We will explain how using classic pseudo-random generators, and via spreading only $O(\log^2 n)$ bits of independent randomness in the network, we can simulate a large string of sufficiently-independent bits of randomness to be able to schedule these BFSs while nodes know their starting times.

3.3 Our Algorithm

We first describe the algorithm assuming that all nodes know a string SR of shared randomness, where $SR[i]$ for $i \in [n-1]$ describes a random value in range $\{1,\dots,\Theta(D)\}$. Later, we explain how to generate SR in $O(D+\log n)$ rounds.

The Distributed FT-BFS Algorithm:

1. Construct a BFS tree $T_0 = BFS(G,s)$ rooted in node s, in $O(D)$ rounds.

2. Number the edges of T_0 by numbers 1 to $n-1$ in an arbitrary way, in $O(D)$ rounds.

3. Make each node v know the numbers of the edges on its shortest path P_v to s in T_0, in $O(D)$ rounds.

4. Let each node v send to each of its neighbors u the numbers of the edges on the shortest path P_v, in $O(D)$ rounds.

5. Divide time into phases of $\Theta(\log n)$ rounds each.

6. Generate the string SR of shared randomness, in $O(D+\log n)$ rounds.

7. Start $BFS(G\setminus\{e\},s)$ for each $e \in T_0$ with a delay of t_e phases, picked uniformly at random from $\{1,2,\dots,D\}$, by setting it equal to $SR[i]$ where i is the edge-number of e. Since SR is publicly known, all nodes know t_e.

8. Run each $BFS(G\setminus\{e\},s)$ at a speed of one hop per phase, following these rules:

   - For each $e'=(v,u)$, each $BFS(G\setminus\{e\},s)$ is permitted to send its token from v to u only if $e \in P_v$. As v knows the edges on P_v, it can identify the permitted BFSs.

   - For each $e \notin P_v$, if v is u's parent in T_0, then u sets its parent in $BFS(G\setminus\{e\},s)$ to be also v. In that case, u proceeds $BFS(G\setminus\{e\},s)$ as if the related BFS token arrived from v to u in phase number $dist_G(s,v)+t_e+1$. Note that node u can do this for all edges $e \notin P_v$ as u knows the numbers of edges $e \in P_v$.

The correctness of the computed BFSs is immediate as for each $BFS(G\setminus\{e\},s)$, each node v will receive the related BFS token in the phase number $dist_{G\setminus\{e\}}(s,v)+t_e$, which since v knows t_e, means v also learns its distance in that BFS, as well as its parent in the BFS. As for the time complexity, each of the first four steps can be performed in $O(D)$ rounds using the standard approach. Steps 4 and 6 are just for explanation and they do not need any communication rounds. We will discuss step 5 later in Lemma 3.2. The execution of the last step finishes within $O(D)$ phases, which is $O(D\log n)$ rounds. This is because each BFS will start at the latest by phase D and after that, it can continue for at most $2D$ additional phases[2].

The only two things that remain to be discussed are as follows: (1) we should show that w.h.p., per phase, at most $O(\log n)$ BFS tokens will need to pass through each edge, (2) we need to explain how to generate the string of shared randomness SR.

Lemma 3.1. *W.h.p., at most $O(\log n)$ BFS tokens need to go through each edge, per phase.*

Proof. We show that w.h.p., in each phase number t and for each edge $e'=(v,u)$, at most $O(\log n)$ BFS tokens will need to go through e' from v to u in phase t.

Note that the only BFSs that are now permitted to send a token from v to u are those trees $BFS(G\setminus\{e\},s)$ for $e \in P_v$, which by definition of P_v, is a total of at most D BFSs. For each such $BFS(G\setminus\{e\},s)$, the token will have to go through e' from v to u in phase t if and only if $dist_{G\setminus\{e\}}(v,s)+t_e = t$. Since t_e is chosen randomly from a range of size D, the probability of this event is at most $1/D$. Hence, over the set of D BFSs for $e \in P_v$, we expect to see at most 1 scheduled to go from v to u in phase t. If the random delay values t_e where completely independent, by an application of Chernoff bound, we would have that this number is at most $O(\log n)$, w.h.p. This will not be exactly true in our case, as we produce SR using a pseudo-random generators, but we will have k-wise independence on the generated string, for $k=\Theta(\log n)$ as we discuss later, and from a result of Schmidt et al. [SSS95], it is known that for this application of Chernoff bound, it suffices to have k-wise independence between the random values, for $k=\Theta(\log n)$. □

Lemma 3.2. *The string of shared randomness SR can be generated and delivered to all nodes in $O(D+\log n)$ rounds.*

[2] It is easy to see that the diameter of each BFS in $G\setminus\{e\}$ is at most $2D$. For general number of faults $f \geq 1$, see [CG84].

Proof. What SR needs to contain is $n-1$ values of randomness, each from a range of size $\Theta(D)$, where these values have $O(\log n)$-wise independent. Suppose that we share $O(\log^2 n)$ random bits that are completely random and independent between all nodes. Note that this can be done in $O(D+\log n)$ rounds, by having the source node s sample these bits using its private randomness, putting them in $O(\log n)$ messages, and then downcasting these messages on the BFS using the standard method. Then, we can feed these bits of true randomness to a pseudo-random number generator. Particularly, each node v feeds these $\Theta(\log^2 n)$ bits into one of the classical k-wise independent, for $k=\Theta(\log n)$, pseudo-randomness constructions (e.g., [AS04, Theorem 15.2.1][3]). This pseudo-random number generator produces n random values, each in range $[\Theta(D)]$, with $\Theta(\log n)$-wise independence between the values. □

4. FAULT-TOLERANT MST

In this section, we explain a distributed algorithm that in $O(D+\sqrt{n}\log n)$ rounds computes an FT-MST structure that is tolerant against failure of each one edge. Recall that to compute an FT-MST structure, we need to compute MST trees in graph $G\setminus\{e\}$ for each edge $e\in G$, which as we discuss next, boils down to computing only n MSTs.

A basic MST tree $T_0 = MST(G)$ can be computed in $O(D+\sqrt{n}\log^* n)$ rounds, using the classic algorithm of Kutten and Peleg [KP95]. Besides this base MST tree, what remains to be computed is $n-1$ additional MST trees, one in each graph $G\setminus\{e\}$ for each edge $e\in T_0$. To compute these MSTs, we essentially need to compute $n-1$ swap edges, as we formalize next.

For a given MST $T_0\subseteq G$ and an edge $e\in T_0$, we call edge $e'=(u,v)$ a *swap edge* for e if adding e' to $T_0\setminus\{e\}$ restores the connectivity of T_0 when e fails. It is easy to see that an edge $e'=(u,v)$ is a swap edge for e iff the edge e is on the unique T_0-path connecting u and v. In other words, edge $e'=(u,v)\in G$ is a swap edge for e if and only if the following conditions are satisfied: (1) edge e appears on the T_0-path connecting the root r to v or to u, and (2) the Least Common Ancestor (LCA) of v and u is above e. See Figure 1.

Given this definition, it is easy to see that when a T_0-edge e fails, to restore the MST, what we need to do is to compute the lightest swap edge for e. Particularly, we have the following fact:

Definition 4.1. *For each edge $e\in T_0$, let $\mathtt{SE}(e)$ be the set of all swap edges for e, by defining $\mathtt{SE}(e)=\{e' \mid (T_0\setminus\{e\})\cup\{e'\}$ is connected$\}$. Moreover, define the best swap edge $\mathtt{SE}^*(e)$ of e to be the lightest edge e' in $\mathtt{SE}(e)$. Note that the tree $T_0\setminus\{e\}\cup\{\mathtt{SE}^*(e)\}$ is an MST of $G\setminus\{e\}$.*

Hence, an FT-MST structure H consists of the edges of the MST tree T_0 along with at most $(n-1)$ best swap edges, one $\mathtt{SE}^*(e)$ for each $e\in T_0$. Thus, to compute FT-MST structures, we need to distributedly compute all $(n-1)$ best swap edges, one $\mathtt{SE}^*(e)$ for each $e\in T_0$.

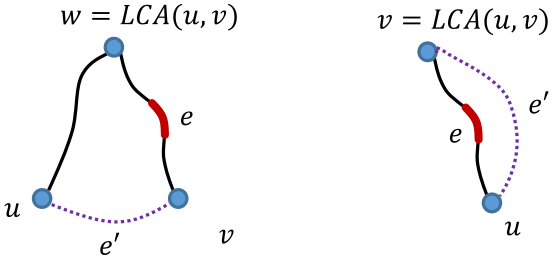

Figure 1: An illustration of a swap edge e' for edge e, in two possible shapes. The solid edges are those of the MST tree T_0, the T_0 edge e is indicated with a red color, and the dashed line indicates e'.

As a warm up, we begin in Section 4.1 by describing an algorithm that computes all these swap edges in $O(D_{MST})$ rounds, where D_{MST} is the diameter of the tree $T_0 = MST(G)$. Notice that D_{MST} can possibly be much larger than the diameter D of graph G. In Section 4.2, we explain how to extend this idea and get an algorithm with round complexity $O(D+\sqrt{n}\log n)$. For that, in a nutshell, we will decompose the tree into $O(\sqrt{n}\log n)$ *segments*, each of diameter $O(\sqrt{n}\log n)$, and we will also use a *virtual skeleton tree* that captures the structure between these segments.

Throughout, we consider the node r with the minimum ID to be the root of T_0 and we consider the edges of T_0 to be directed away from the root r. A vertex v is said to be *above* an edge e in the tree T_0 if it is closer (in hops) to the root r in T_0.

4.1 Warm Up: FT-MST in $O(D_{MST})$ rounds

To compute the swap edges, we make each vertex v know the T_0 path $\pi(r,v)$ connecting root r to v and moreover, we make each two neighboring vertices exchange these path descriptions. Note that this can be easily done in $O(D_{MST})$ rounds, using standard methods: computing the path $\pi(r,v)$ can be done by downcasting the parent IDs from the root to the leaves. Then, each two neighbors need to exchange at most $O(D_{MST})$ path edges, which clearly can be done in $O(D_{MST})$ rounds. At the end, each two neighboring vertices u and v connected via edge e' can know all T_0-edges e for which e' is a swap edge for e. Particularly, v and u compute their LCA w in T_0 and then edge e' is a swap edge for each T_0-edge $e\in\pi(r,v)\cup\pi(r,u)$ that is below w. That is, if e' is on the unique T_0-path connecting v and u.

The algorithm proceeds as follows: each vertex v evaluates which of its incident edges e' are swap edges for each T_0-edge $e\in\pi(r,v)$. Specifically, vertex v defines its swap proposal $\mathtt{SE}^*(e,v)$ for each edge $e\in\pi(r,v)$ to be the lightest edge e' incident on v that is a swap edge for e.

Next, the edges $\mathtt{SE}^*(e,v)$ along with their weights are upcasted towards the head endpoint y of the edge $e=(x,y)$ in the following pipeline manner: Letting $\pi(r,v)=[e_1,\ldots,e_k]$, each vertex v maintains a list of edges $\mathtt{SE}^*(e_1,v),\ldots,\mathtt{SE}^*(e_k,v)$, where each such edge $\mathtt{SE}^*(e_i,v)$ is the current best swap edge for e_i that v knows. Initially, the $\mathtt{SE}^*(e_i,v)$ edges are those incident to v. During the upcast, upon receiving values from its descendants, these values may correspond to edges with endpoint in the subtree of v. The vertex v keeps on forward-

[3] We note that [AS04, Theorem 15.2.1] only describes the construction for GF(2) which yields k-wise independent bit. However, as also described in [Cou, Section 3], the construction readily extends to $GF(p)$, for any prime number $p\in \text{poly}(n)$. Furthermore, when desiring random delays in range $[\Theta(D)]$, we can pick them from a range $\{1,\ldots,p\}$ for a prime $p\in[D,10D]$, which exists by Bertrand's postulate.

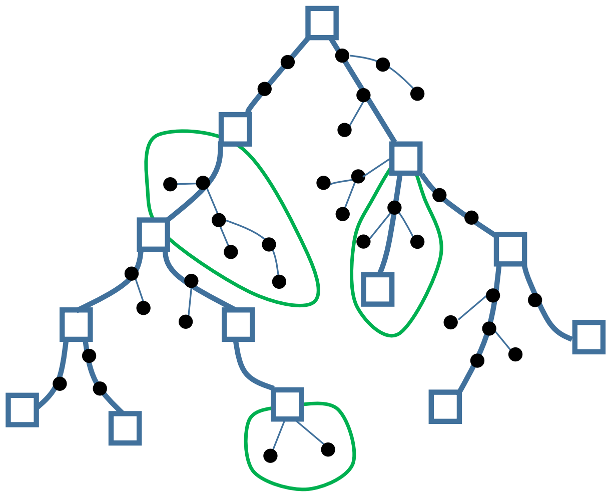

Figure 2: A schematic depiction of the decomposition of the MST and its skeleton tree. The marked nodes are depicted as blue boxes, while small black circles indicate non-marked nodes. One can obtain the *skeleton tree* by erasing the non-marked nodes, i.e., keeping only the thicker blue lines between the boxes. The three green curves show three sample segments; for simplicity, we do not highlight all the segments in the picture.

ing to its parent in T_0 arbitrary stored items that have not been upcast so far; upon receiving an edge $\texttt{SE}^*(e_i, u)$ from a descendant u, the vertex v updates its $\texttt{SE}^*(e, v)$ values only if the swap edge suggested by u is better, i.e., of smaller weight. Since overall every vertex v computes the minimum of $O(D_{MST})$ functions in its subtree T_v, this piplelined upcast finishes in $O(D_{MST})$ rounds (cf., [Pel00]).

By the end of this process, we get that every vertex v holds the best swap edge $\texttt{SE}^*(e_v)$ where e_v is the incoming edge to v in T_0. By rewinding the tape of the communication, the endpoints of the selected swap edges $\texttt{SE}^*(e_v)$ for every v are notified, learning that their edge is the best swap edge for e_v. They can then add these edges to the FT-MST structure H. This completes the description for our warm up $O(D_{MST})$ round algorithm.

4.2 FT-MST in $O(D + \sqrt{n}\log n)$ rounds

We start the description of the algorithm by defining a decomposition of the MST tree into a number of *segments* and a skeleton tree capturing this decomposition. For simplicity, we then first explain how to use these structures to compute the best swap edges, and only later come back to the description of distributed construction of these structures.

Pre-processing—Segments and Skeleton Tree: To define the tree decomposition into segments, we mark a number of nodes on the MST tree T_0 such that these marked nodes satisfy the following three properties: (P1) the root r is a marked node and for each node $v \neq r$, there is a marked node ancestor u of v such that $\texttt{dist}_{T_0}(u, v) \leq O(\sqrt{n}\log n)$, (P2) for each two marked nodes u and v, their least common ancestor is also a marked node, and (P3) the number of the marked nodes is at most $O(\sqrt{n}\log n)$. See Figure 2.

Given these marked nodes, we decompose the MST tree T_0 into segments as follows: for each node v, define its lowest ancestor marked node $LAM(v)$ and its highest descendant marked node $HDM(v)$, if such exists, and $\emptyset$ otherwise. Each segment is defined to include the set of all nodes with the same pair LAM and HDM. Note that these segments are not vertex-disjoint, as they overlap in marked nodes, but they are edge-disjoint, and therefore, we can perform independent communications in different segments in parallel.

Due to properties (P1) and (P3), we have $O(\sqrt{n}\log n)$ segments, each with a diameter of $O(\sqrt{n}\log n)$ hops. Having these segments, we further define a virtual *skeleton tree* T_S whose vertices are the marked nodes and a marked node v is the T_S-parent of marked node u iff $v = LAM(u)$. Putting it in intuitive (and imprecise) words, the skeleton tree can be obtained simply by erasing all non marked nodes, while paths going through them are maintained and turn into virtual edges. See Figure 2.

In the following, we assume that vertices have the following knowledge about these structures:

(K1) each vertex v knows the identifier of its segment, namely $LAM(v)$ and $HDM(v)$,

(K2) each vertex v knows the identifiers of the T_0-edges on the unique path from v to each of these two vertices $LAM(v)$ and $HDM(v)$, and

(K3) all vertices know the full topology (vertices and edges) of the skeleton tree T_S.

We will explain later in Section 4.3 that all this knowledge can be attained in $O(D + \sqrt{n}\log n)$ rounds. For now, we assume this knowledge as given, and proceed to explain how to compute the best swap edges, using these structures.

Classifying swap edges to short, mid, and long ranges: Given a T_0-edge e and a swap edge of it $e' = (u', v') \in \texttt{SE}(e)$, we call e' a *short-range*, *mid-range*, or *long-range* swap for e if and only if, respectively, two, one or zero of the endpoints u' and v' of e' are in the same segment as e. See Figure 3 for some examples.

The *best short-range* swap for e is the *lightest* short-range edge $e' \in \texttt{SE}(e)$; the *best mid-range* and *best long-range* swaps for e are defined analogously. We compute the *best short-range*, the *best mid-range*, and the *best long-range* swaps separately. At the end, we set the best swap edge of each T_0-edge e to be simply the lightest of its best swaps in these three categories.

Computing short-range swap edges: The computations of short-range swaps in different segments happen independently and in parallel, each via communicating only on the edges of that segment. Before starting that computation of these swap edges, we make every two neighbors exchange their segment identifiers.

We focus on one segment S_i and the T_0-subtree T_i induced by vertices of S_i. Our goal is to compute the best short-range swap edge $\texttt{SE}^*_{short}(e)$ for each tree edge $e = (x, y) \in T_i$. To do that, we employ the algorithm described in the previous subsection, restricted to tree T_i, as follows: Assume that all edges of T_i are directed away from the root r_i. We let every two neighboring vertices v and u in the same segment, connected via edge $e' = (u, v)$, exchange items (K2) of their knowledge, i.e., their T_0-paths to $LAM(v) = LAM(u)$. Notice that this can be performed in $O(\sqrt{n}\log n)$ rounds, as

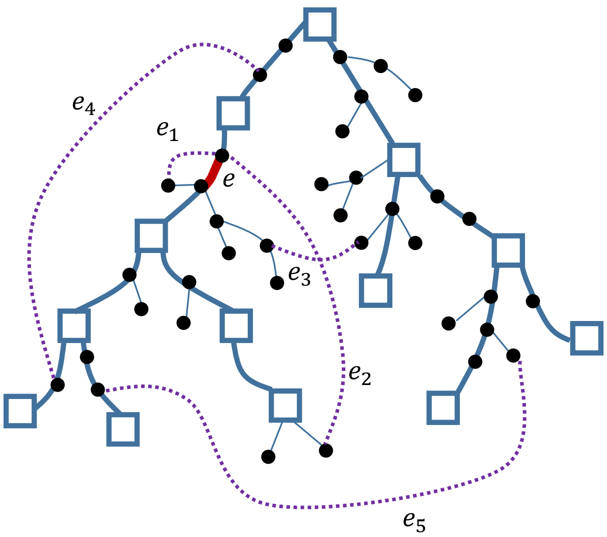

Figure 3: For the T_0-edge e indicated with a red color, the edge e_1 is a short-range swap, the edges e_2 and e_3 are mid-range swaps, and the edges e_4 and e_5 are long-range swaps.

each of these paths has length $O(\sqrt{n}\log n)$. Using this information, nodes v and u can compute the unique T_0-path connecting v and u. Then, v and u include edge e' in their computations of swap proposals for all T_0-edges on this path. The rest of the computation for finding the best short-range swap edge is a simple upcast, exactly as described in the previous subsection.

Computing mid-range swap edges: Similar to the short-range swaps, the mid-range swaps of different segments happen independently and in parallel, after nodes know the segment identifiers of their neighbors. Let us focus on one segment S_i. Consider a node $v \in S_i$ and a neighbor of it u in a different segment S_j, connected to v via edge $e' = (v, u)$. We say that segments S_i and S_j are *dependent* if either all nodes of S_i are in the T_0-subtree below (the lowest marked node of) S_j, or all nodes of S_j are in the T_0-subtree below S_i. Otherwise, we say these two segments are *independent*. We next examine these two cases separately:

- **Independent Segments S_i and S_j:** In this case, the LCA of v and u is above $LAM(v)$ and e' is a mid-range swap edge for all edges on the T_0-path connecting v to its lowest ancestor marked ndoe $LAM(v)$.
- **Dependent Segments S_i and S_j:** This case breaks into two possibilities: either S_j is a descendant of S_i or S_i is a descendant of S_j. In the former possibility, the LCA of v and u is in S_i and e' is a mid-range swap edge for all edges on the T_0-path connecting v to $HDM(v)$. In the latter possibility, the LCA of v and u is in S_j and e' is a mid-range swap edge for all edges on the T_0-path connecting v to $LAM(v)$.

Since v knows the segment identifier of u as well as the full skeleton tree structure T_S, node v can identify in which of the above cases we are. Furthermore, since v knows its paths to its own $LAM(v)$ and $HDM(v)$, node v knows which T_0-edges of segment S_i can use e' as a mid-range swap edge. To have the best mid-range computed, we have each such node v put its proposals for each of the relevant T_0-edges e. We then perform one upcast in $T_0[S_i]$ towards $LAM(v) = LAM(S_i)$, thus obtaining the best mid-range swaps for which the segment endpoint of the swap edge is below the swapped edge. Afterward, we also perform a downcast towards $HDM(S_i)$, thus obtaining the best mid-range swaps for which the segment endpoint of the swap edge is above the swapped edge. Each of these upcast and downcast is essentially identical to, or reverse of, the upcast in the algorithm of the previous subsection. Hence, after at most $O(\sqrt{n}\log n)$ rounds, all edges know their best mid-range swaps.

Computing long-range swap edges: For this part, we use the skeleton tree. We consider $O(\sqrt{n}\log n)$ minimization problems, one for each skeleton edge. For each edge $e' = (u, v) \in G$, the two endpoints u and v can easily find the skeleton edges e for which e' is a swap edge, that is, the skeleton edges e which reside on the skeleton path connecting the closest marked nodes of the segments of u and v. Then, v and u include e' in their proposal for swap edges of e. We perform a pipelined upcast for these $O(\sqrt{n}\log n)$ minimization problems on the BFS tree of the graph G. For each problem, each node always maintains the lightest solution for that problem seen so far, and passes it to its parent in the BFS tree. A final downcast then delivers these solutions to all nodes. At the end, since for each physical edge $e'' \in T_0$, the endpoints know to which virtual skeleton edge e does edge e'' belong, they can take the solution of the minimization of e as the best long range swap for e''. Note that in this manner, we are treating all the physical edges that are on the same virtual skeleton edge essentially the same, for the purpose of their long-range swaps. We next formally argue that this indeed is correct, which means we compute the best long-range swaps for all T_0-edges.

PROPOSITION 4.2. *Let e_1, e_2 be two T_0-edges in the same segment and suppose that e'_1 and e'_2 are the best long-range swap edges for e_1 and e_2, respectively. Then, the two best long-range swap edges are the same and we have $e'_1 = e'_2$.*

PROOF. Notice that since e_1 and e_2 have long range swaps, they both must be on the main path of their segment which connects their lowest ancestor marked node w_1 to their highest descendant marked node w_2. Now, since e'_1 is long range swap edges for e_1, it must be the case the T_0-path connecting w_1 and w_2 is a subpath of the T_0-path connecting the two endpoints of e'_1. A similar thing is also true for e'_2. Hence, e'_1 and e'_2 are both long range swaps for both of e_1 and e_2. Since e'_1 and e'_2 are the best long-range swap edges for e_1 and e_2, by the uniqueness of the edge weights, we then get that $e'_1 = e'_2$. □

Putting things together: At the end, for each T_0-edge e, the best swap edge e' is the lightest of its best short-range, mid-range, and long-range swaps. Since the endpoints of e know the best of each category, they can easily find the best swap edge. Finally, we can reverse the direction and time-order of all these communications, i.e., essentially rewinding the tape of communications backwards, and thus let the endpoints of each best swap edge e' know that it is the best swap edge for some T_0-edge e. The endpoints of e' can then add e' to the FT-MST structure H. This completes the

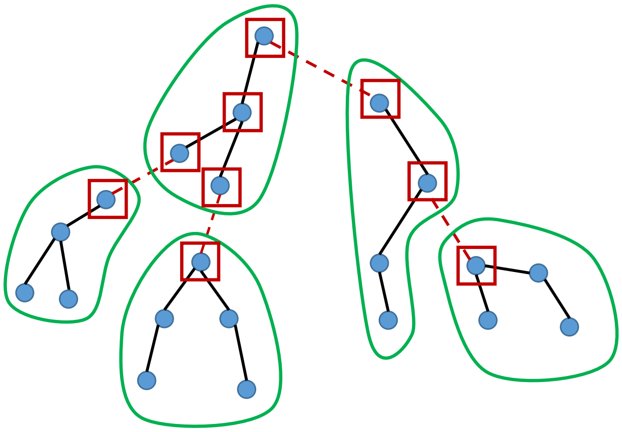

Figure 4: The components in the decomposition are marked with green curves, the sampled edges are depicted as dashed red edges, and red boxes indicate marked nodes, i.e., the nodes of virtual skeleton tree.

description of how we compute the swap edges, using the skeleton structure and the segments. We next go back to explain how we compute these structures in $O(D+\sqrt{n}\log n)$ rounds, as claimed.

4.3 Constructing the Segments and the Skeleton Tree

Here, we explain how to identify the marked nodes such that they satisfy the properties (P1) to (P3), stated in Section 4.2. We then explain how to use these marks to identify the segments and the skeleton tree and let each node v have the knowledge items (K1) to (K3), stated in Section 4.2.

First, define a random subset of MST edges $R \subseteq T_0$ by including each T_0-edge in E independently with probability $1/\sqrt{n}$. First, we show that w.h.p. $T_0 \setminus R$ has $O(\sqrt{n}\log n)$ connected components, each with a diameter of at most $O(\sqrt{n}\log n)$. See Figure 4.

Lemma 4.3. *W.h.p., $T_0 \setminus R$ has $O(\sqrt{n}\log n)$ connected components, each with a diameter of at most $O(\sqrt{n}\log n)$.*

Proof. Since every edge in T_0 is added to R with probability $p = 1/\sqrt{n}$, by Chernoff bound, we get that $|R| = O(\sqrt{n}\cdot\log n)$, with high probability. Hence, the number of connected components of $T_0 \setminus R$ is $O(\sqrt{n}\cdot\log n)$.

We next bound the diameter of the tree induced by the vertices of each component C_i in $T_0 \setminus R$. Consider all pairs $\mathcal{P}$ of vertices u and v whose distance in the tree T_0 is exactly $c\cdot\log n\cdot\sqrt{n}$. That is letting $\pi(u,v)$ be the $u-v$ path in T_0, then $\mathcal{P} = \{\langle u,v\rangle \mid |\pi(u,v)| = c\cdot\log n\cdot\sqrt{n}\}$. We claim that w.h.p. $R \cap E(\pi(u,v)) \neq \emptyset$ for every $\langle u,v\rangle \in \mathcal{P}$. To see this, consider one pair u and v in $\mathcal{P}$. The probability that none of the edges on $\pi(u,v)$ has been sampled into R is $(1-1/\sqrt{n})^{c\cdot\log n\cdot\sqrt{n}} = 1/n^c$. Hence, by union bound over at most $O(n^2)$ pairs, we get that w.h.p the tree paths of all pairs in $\mathcal{P}$ intersect R. Therefore, w.h.p., the diameter of every component of $T \setminus R$ is $O(\sqrt{n}\log n)$. □

We now proceed to use these components for defining the marked nodes, the segments, and the skeleton. First, we start with orienting the tree of each component downward, with respect to the global root. We then use these oriented component trees to define the marked nodes. At the end, we use these marked nodes to construct the segments and the skeleton tree.

Pre-processing Step 1—Defining a rooted subtree $T_i' \subseteq T$ for each component C_i: First, remove the edges of R and perform a minimum ID flooding on edges of $T_0 \setminus R$. Thus, within $O(\sqrt{n}\log n)$ rounds, all nodes of each component C_i of $T_0 \setminus R$ have the smallest ID of that component.

We next wish to define a root r_i' for each component C_i. In particular, assuming that all edges of T_0 are directed away from the global root r, the root r_i' of the component C_i is the vertex with incoming degree zero in the subtree T_i' induced on the vertices of C_i. To let every vertex know the root of its component, we define an intermediate virtual tree T' obtained by contracting every component C_i of $T_0 \setminus R$ into a single vertex. By exchanging the component IDs with the neighbors, the edges of T' can become global knowledge within $O(D+\sqrt{n}\log n)$ rounds, w.h.p. This is because there are $O(\sqrt{n}\log n)$ components (w.h.p.) and the T_0-edges connecting vertices of different components, which are those in R, can be communicated to the entire graph using an upcast on the BFS tree, in $O(D+\sqrt{n}\log n)$ rounds. Hence, by rooting the intermediate virtual tree T' at the component of the root vertex r, each vertex can know the root of its own component, all in $O(D+\sqrt{n}\log n)$ rounds. Then, by performing a downcast from the root of each component on the edges of $T_0 \setminus R$, in $O(D+\sqrt{n}\log n)$ rounds, we get that each component of $T_0 \setminus R$ is directed outwards from its root and each node knows its parent—that is, its unique incoming neighbor—in this orientation.

Pre-processing Step 2—Marking nodes, inside components C_i: First, mark each vertex that is incident on a sampled edge in R, as well as the root r of T_0. Then, we start an upcast in each component C_i, on the tree of $T_0[C_i]$ from the leaves towards the root. During this process, we identify and mark nodes of $T_0[C_i]$ that are LCA of two of these sampled edges (i.e., their marked nodes). Particularly, any leaf v that is a marked node sends the identifier of its sampled edge to its parent, indicating that v is marked. Then, when the upcast reaches a node u, (1) if u itself is incident on a sampled edge, then u is marked and it passes the ID of one such sampled edge to its parent, (2) if u is not marked and it receives only one sampled edge from its children, it simply passes this sampled edge to its parent, and (3) if node u is not marked but it receives two or more sampled edges from its children, then u becomes marked and it passes only one of these sampled edges to its parent.

Lemma 4.4. *The set of marked nodes satisfy the following properties: (P1) the root r is marked and for each node $v \neq r$, there is a marked node ancestor u of v such that $\mathtt{dist}_{T_0}(u,v) \leq O(\sqrt{n}\log n)$, (P2) for each two marked nodes u and v, their least common ancestor is also a marked node, and (P3) the number of the marked nodes is $O(\sqrt{n}\log n)$.*

Proof. Property (P1) follows immediately because each component has depth at most $O(\sqrt{n}\log n)$ and the root of each component is marked, as either it is the root of T_0, or it is incident on a sampled edge in R.

For property (P2), consider two marked nodes u and v. If v is a descendant or ancestor of u, then their LCA is u

or v, respectively, which is marked. So suppose that v is not a descendant or ancestor of u. Suppose that u and v are in the same connected component. Then, all edges on the T_0-path connecting these two nodes will carry marks, while we perform the upcast explained above. This means that their LCA receives at least two sampled edges from its children and thus it becomes marked, too. Now suppose that v and u are in two different component. Then, consider the component that includes the LCA w of these two nodes, and the parts of the paths from w to v and from w to u that are in this component. Clearly, these paths end at two different sampled edges. Thus, these endpoints are marked. This means node w is in fact the LCA of two marked nodes in the same component. As we argued above, in this case, we know that node w is marked.

Finally, for (P3), note that the initial set of marked nodes is those vertices that are incident on sampled edges, and since $|R| = O(\sqrt{n} \log n)$, the set of initial marked nodes has size $O(\sqrt{n} \log n)$. Now, as we perform the upcast of marking, each new node w that we mark receives at least two sampled edges from its children, but it passes only one sampled edge to its parent. Hence, we can charge the marking of w to one of the sampled edges that it received but did not pass up to its parent. This shows that the number of additional marks that we introduce during the marking upcast is one per sampled edge, which is at most $O(\sqrt{n} \log n)$. □

Pre-processing Step 3—Defining the Segments and the Skeleton tree T_S: To define the edges of the skeleton tree T_S, we perform an upcast inside each component, where each marked node v that is not the root of a component sends its ID up to its parent. We continue this upcast and the ID gets passed on to the parent until it reaches the lowest ancestor marked node w. Then, w knows that it is the parent of v in the skeleton tree T_S, and there is a virtual edge between w and v in T_S. Since there are $O(\sqrt{n} \log n)$ marked nodes, there are also $O(\sqrt{n} \log n)$ such virtual edges in the skeleton tree. We broadcast all these virtual edges to all nodes, in $O(D + \sqrt{n} \log n)$ rounds. This ensures that all nodes know the skeleton tree, i.e., it satisfies knowledge item (K3) in Section 4.2. Then, by performing a downcast from each marked node, and after that also an upcast from each marked node, and stopping these downcast and upcast once they reach the first next marked node, we can make each nodes v know the identifiers of its lowest ancestor marked node $LAM(v)$ and its highest descendant marked node $HDM(v)$. This is the segment identifier of v and thus it satisfies knowledge item (K1). Finally, by repeating these upcasts and downcast while also carrying the names of all visited edges, we can make each node v know the IDs of the edges on its paths to $LAM(v)$ and its highest descendant marked node $HDM(v)$, thus satisfying (K2). This finishes our description of the knowledge items (K1) to (K3), and hence also the construction of the segments and the skeleton tree.

5. REFERENCES

[AS04] Noga Alon and Joel H Spencer. *The probabilistic method*. John Wiley & Sons, 2004.

[BGLP14] Davide Bilò, Luciano Gualà, Stefano Leucci, and Guido Proietti. Fault-tolerant approximate shortest-path trees. In *Algorithms-ESA 2014*, pages 137–148. Springer, 2014.

[BK13] Surender Baswana and Neelesh Khanna. Approximate shortest paths avoiding a failed vertex: Near optimal data structures for undirected unweighted graphs. *Algorithmica*, 66(1):18–50, 2013.

[CG84] FRK Chung and MR Garey. Diameter bounds for altered graphs. *Journal of Graph Theory*, 8(4):511–534, 1984.

[Cou] Lecture notes on k-wise uniform (randomness) generators. http://pages.cs.wisc.edu/~dieter/Courses/2013s-CS880/Scribes/PDF/lecture04.pdf. Accessed: 2015-02.

[CP10] Shiri Chechik and David Peleg. Rigid and competitive fault tolerance for logical information structures in networks. In *Electrical and Electronics Engineers in Israel (IEEEI), 2010 IEEE 26th Convention of*, pages 000024–000025. IEEE, 2010.

[DLPP13] Ajoy K Datta, Lawrence L Larmore, Linda Pagli, and Giuseppe Prencipe. Linear time distributed swap edge algorithms. In *Algorithms and Complexity*, pages 122–133. Springer, 2013.

[DSHK$^+$11] 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. In *Proc. of the Symp. on Theory of Comp. (STOC)*, pages 363–372, 2011.

[Elk04] Michael Elkin. Unconditional lower bounds on the time-approximation tradeoffs for the distributed minimum spanning tree problem. In *Proc. of the Symp. on Theory of Comp. (STOC)*, pages 331–340, 2004.

[FEP$^+$12] Paola Flocchini, T Enriquez, Linda Pagli, Giuseppe Prencipe, and Nicola Santoro. Distributed minimum spanning tree maintenance for transient node failures. *Computers, IEEE Transactions on*, 61(3):408–414, 2012.

[Gha15] Mohsen Ghaffari. Near-optimal scheduling of distributed algorithms. In *the Proc. of the Int'l Symp. on Princ. of Dist. Comp. (PODC)*, pages 3–12, 2015.

[GL09] Zvi Gotthilf and Moshe Lewenstein. Improved algorithms for the k simple shortest paths and the replacement paths problems. *Information Processing Letters*, 109(7):352–355, 2009.

[GSW11] Beat Gfeller, Nicola Santoro, and Peter Widmayer. A distributed algorithm for finding all best swap edges of a minimum-diameter spanning tree. *Dependable and Secure Computing, IEEE Transactions on*, 8(1):1–12, 2011.

[GW12] Fabrizio Grandoni and Virginia Vassilevska Williams. Improved distance sensitivity oracles via fast single-source replacement paths. In *Foundations of Computer Science*

(FOCS), 2012 IEEE 53rd Annual Symposium on, pages 748–757. IEEE, 2012.

[HS01] John Hershberger and Subhash Suri. Vickrey prices and shortest paths: What is an edge worth? In *Foundations of Computer Science, 2001. Proceedings. 42nd IEEE Symposium on*, pages 252–259. IEEE, 2001.

[HW12] Stephan Holzer and Roger Wattenhofer. Optimal distributed all pairs shortest paths and applications. In *the Proc. of the Int'l Symp. on Princ. of Dist. Comp. (PODC)*, pages 355–364, 2012.

[KP95] Shay Kutten and David Peleg. Fast distributed construction of k-dominating sets and applications. In *the Proc. of the Int'l Symp. on Princ. of Dist. Comp. (PODC)*, pages 238–251, 1995.

[LMR94] Frank Thomson Leighton, Bruce M Maggs, and Satish B Rao. Packet routing and job-shop scheduling in O(congestion+ dilation) steps. *Combinatorica*, 14(2):167–186, 1994.

[MMG89] Kavindra Malik, AK Mittal, and Santosh K Gupta. The k most vital arcs in the shortest path problem. *Operations Research Letters*, 8(4):223–227, 1989.

[NR99] Noam Nisan and Amir Ronen. Algorithmic mechanism design. In *Proceedings of the thirty-first annual ACM symposium on Theory of computing*, pages 129–140. ACM, 1999.

[NSW97] Enrico Nardelli, Ulrike Stege, and Peter Widmayer. *Low-cost Fault-tolerant Spanning Graphs for Point Aets in the Euclidean Plane.* 1997.

[Pel00] David Peleg. *Distributed Computing: A Locality-sensitive Approach.* Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 2000.

[Pet05] Seth Pettie. Sensitivity analysis of minimum spanning trees in sub-inverse-ackermann time. *Algorithms and Computation*, pages 964–973, 2005.

[PP13] Merav Parter and David Peleg. Sparse fault-tolerant BFS trees. In *the Proceedings of the Annual European Symposium on Algorithms*, pages 779–790, 2013.

[PP14] Merav Parter and David Peleg. Fault tolerant approximate bfs structures. In *Proceedings of the Twenty-Fifth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1073–1092. SIAM, 2014.

[PR99] David Peleg and Vitaly Rubinovich. A near-tight lower bound on the time complexity of distributed MST construction. In *Proc. of the Symp. on Found. of Comp. Sci. (FOCS)*, pages 253–, 1999.

[RZ12] Liam Roditty and Uri Zwick. Replacement paths and k simple shortest paths in unweighted directed graphs. *ACM Transactions on Algorithms (TALG)*, 8(4):33, 2012.

[SSS95] Jeanette P Schmidt, Alan Siegel, and Aravind Srinivasan. Chernoff-hoeffding bounds for applications with limited independence. *SIAM Journal on Discrete Mathematics*, 8(2):223–250, 1995.

[Tar82] Robert Endre Tarjan. Sensitivity Analysis of Minimum Spanning Trees and Shortest Path Trees. *Inf. Process. Lett.*, 14(1):30–33, 1982.

[Wil11] Virginia Vassilevska Williams. Faster replacement paths. In *Proceedings of the twenty-second annual ACM-SIAM symposium on Discrete Algorithms*, pages 1337–1346. SIAM, 2011.

[WY13] Oren Weimann and Raphael Yuster. Replacement paths and distance sensitivity oracles via fast matrix multiplication. *ACM Transactions on Algorithms (TALG)*, 9(2):14, 2013.