algorithms

MDPI

Article

Maintaining Minimum Spanning Trees in Dynamic Distributed Networks: Non-Tree-Edge Tracking with Predictive Scheduling

Asrar U. Haque

Department of Computer Science, College of Computer Sciences and Information Technology, King Faisal University, Al-Ahsa 31982, Saudi Arabia; ahaque@kfu.edu.sa

Abstract

Spanning structures that must be maintained, not merely rebuilt, underlie sensor fabrics, peer-to-peer overlays, and software-defined networks. We give a complete treatment of distributed minimum-spanning-tree (MST) construction and maintenance built on non-tree-edge (NTE) tracking, which certifies the edges excluded from the tree rather than growing tree fragments. First, we provide the full specification and correctness proofs of the static NTE algorithm—previously available only in outline—achieving $O(m)$ messages of $O(\log n)$ bits in $O(d)$ rounds with only three message types. Second, we prove an exact maintenance algorithm for edge-weight updates: three of four update cases resolve in $O(1)$, and the fourth is resolved by ExactSwap within a message bound that experiments show to be tight (median ratio 1.00); across 221,874 update records, the median affected set is 2, independent of scale ($n = 500$–$20{,}000$). Third, we establish a separation result: verifying a replacement edge distributively provably requires examining the entire search region, an obstruction with no centralized analogue. Fourth, we show that prediction is best used for scheduling: under contention, prioritizing predicted-cheap updates improves mean time-to-resolution by 2.1–3.8× over FIFO, matching an oracle, with a linear model sufficing.

Keywords: distributed algorithms; minimum spanning tree; dynamic graph algorithms; CONGEST model; learning-augmented algorithms; scheduling; sensor networks; overlay networks; Internet of Things

1. Introduction

The infrastructures that define large-scale computing today are wide-area, fully distributed, and in constant flux. Sensor fabrics in the Internet of Things coordinate thousands of energy-constrained nodes whose links degrade and whose members die [1,2]; peer-to-peer and overlay networks reorganize continually under churn [3]; federated-learning deployments train models across participant sets that grow and shrink between rounds [4,5]; and software-defined fabrics must repair multicast and control structures when links fail [6]. All of these depend on a common substrate: a low-cost spanning structure over the current participants. Building that structure once is a solved problem; *keeping it correct and minimal as the infrastructure changes*—without re-running a global computation on every event—is not, and at the scales above, the difference between rebuilding and repairing is the difference between an infrastructure that spends its bandwidth and energy on its mission and one that spends them on self-maintenance.

Distributed construction of a minimum spanning tree (MST), the canonical such structure, is a classical problem in distributed computing, with algorithms dating to Gallager, Humblet and Spira (GHS) [7] and a substantial line of subsequent work improving message

Check for updates

Academic Editor: Frank Werner

Received: 12 August 2026
Revised: 2 September 2026
Accepted: 5 September 2026
Published: 8 September 2026

Copyright: © 2026 by the author. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license.

and time complexity [8–12]. Almost all of this work addresses *static construction*: given a fixed graph, build its MST once. In practice, the networks these algorithms run over rarely stay fixed. Peer-to-peer overlays experience churn as participants join and leave; wireless sensor networks lose nodes to battery depletion and suffer channel-quality fluctuation from interference and mobility; software-defined network fabrics reprovision links and route around failures. An MST built once becomes stale as soon as the underlying graph changes, and the naive response, recomputing from scratch, discards everything the network already knew and pays the full construction cost again.

How often this matters is an empirical question, and the answer shapes what a dynamic algorithm should optimize for. Real link costs change on two very different timescales, and only discrete structural events—not short-term measurement jitter—alter the optimal tree; our update model makes this distinction explicit and grounds it in measurement literature.

The applications motivating this distinction are concrete and, in two cases, the author's own prior work. Ragab and Haque [13] applied an early, structurally-outlined version of the algorithm formalized in full here to eliminate duplicate packet delivery in δ, μ-bounded multicast overlay clusters for peer-to-peer video streaming—an application where peer churn and link-quality fluctuation are the norm. Khan and Haque [14] applied MST-based aggregation to extend the lifetime of power-limited sensor networks, where node death from battery depletion is a discrete event of exactly the kind our dynamic extension is built to handle efficiently—a motivation independently echoed by the Dynamic GHS of Díaz et al. [15]. Beyond these, the same structure recurs in SDN-controlled multicast tree maintenance under link failure, in decentralized federated learning's communication-efficient client overlays under participant churn, and, as very recent work confirms [16], in large-scale dynamic graph analytics generally, where full MST recomputation is explicitly noted to lack practical efficiency under frequent updates.

Learning-augmented algorithms offer a further tool for problems of this shape. Lykouris and Vassilvitskii [17] and subsequent work [18] give online algorithms access to a machine-learned prediction, with guarantees that improve when the prediction is accurate and degrade no worse than a prediction-free baseline when it is not. Van den Brand et al. [19] extend this framework to dynamic graph algorithms (reachability, shortest paths, matching) in a centralized model. Whether an analogous guarantee is available for distributed MST maintenance is a natural question we investigate; the answer, established in Section 5.5, is more nuanced than a straightforward "yes."

Against this background, our contributions are as follows. *First*, a full specification and correctness proof of the NTE (non-tree-edge tracking) distributed MST algorithm (Section 4), first outlined in the author's earlier conference paper [13] and originating in the author's dissertation [20]. The conference presentation established the two-phase structure, the module decomposition and message types, and proofs of Path Optimality and graph orientation, but stated the complexity without proof; we supply the complete module-level specification (full slate design and per-module pseudocode), the Cycle Theorem, Theorem NTE with its supporting lemmas, and the complexity proofs.

Second, a provably correct, non-learned algorithm for maintaining the MST under a sequence of edge-weight changes (Section 5), achieving $O(d)$ rounds and $O(\sum_v \delta_{\mathrm{NTE}}(v))$ messages, summed over the smaller side of the induced cut, for the one update case that can force a structural change—a bound tightened, via a structural observation about spanning-tree cuts, well below the naive degree-based estimate.

Third, a separation between the distributed and centralized settings (Section 5.5, Proposition 4): in the distributed model, verifying the true minimum-weight replacement edge provably requires examining the full search region—no distance-based local certificate can

confirm a partial search using only information visible within a bounded region. This obstruction has no analogue in centralized dynamic-MST maintenance, where global visibility makes the question moot. It cleanly delimits what any distributed algorithm, learned or not, can achieve for this problem, and directly motivates the design of our fourth contribution.

Fourth, a scheduling-aware use of prediction built on top of the proven algorithm (Section 6): since the true replacement edge cannot be found via a shortcut, prediction instead informs *when* the unavoidable computation is carried out, letting systems that tolerate bounded staleness prioritize cheap and small updates over expensive ones without ever affecting correctness (Proposition 5).

The remainder of the paper is organized as follows. Section 2 surveys related work; Section 3 defines the model; Section 4 gives the static algorithm; Section 5 the dynamic extension and the separation result; Section 6 the scheduling layer; Section 7 the experimental evaluation; and Section 9 concludes.

2. Related Work

2.1. Classical Distributed MST

The foundational result is Gallager, Humblet and Spira's distributed MST algorithm [7], which grows fragments and merges them by a minimum-outgoing-edge rule, achieving $O(n \log n)$ time and $O(m + n \log n)$ messages; Chin and Ting [21] improved its time to almost linear. Awerbuch [8] improved the time to $O(n)$, optimal with respect to n, retaining near-linear message complexity; Garay, Kutten and Peleg [9] then argued that the network diameter is the more sensitive parameter and gave the first diameter-sensitive sublinear-time algorithm, running in $O(d + n^{0.614})$ rounds (throughout, d denotes the network diameter; see Section 3). Faloutsos and Molle [10] revisited these algorithms and clarified correctness subtleties. Lotker, Patt-Shamir, Pavlov and Peleg [11] gave an $O(\log \log n)$-round algorithm for fully connected (overlay-style) networks. All of these are *tree-edge-tracking*: non-tree edges are discarded once superseded. The present paper inverts this, tracking the non-tree edges via the Path Optimality criterion (Theorem 2); the approach was outlined in the author's conference paper [13] and is specified and proved in full here.

2.2. Modern CONGEST-Model MST

The CONGEST model [22] is the standard cost model for network-scale distributed algorithms: nodes compute in synchronous rounds and each link carries at most $O(\log n)$ bits per round, so bandwidth, not local computation, is the scarce resource, matching the economics of the infrastructures in Section 1 (Section 3 gives the formal definition). Within it, Kutten and Peleg [23] achieved $O(d + \sqrt{n} \log^* n)$ rounds, near-optimal in view of the $\Omega(d + \sqrt{n}/\log n)$ lower bound of Peleg and Rubinovich [24], which holds even for networks of diameter $d = \Omega(\log n)$; Pandurangan, Robinson and Scquizzato [12] attained near-optimality in *both* rounds and messages simultaneously, and Elkin [25] gave a simple deterministic algorithm with near-optimal time and message complexity; Khan and Pandurangan [26] gave fast distributed approximation algorithms. Parallel (as opposed to message-passing distributed) constructions form a neighbouring line (e.g., Huang's fully pipelined MST constructor [27]) analyzed under parallel cost models rather than the CONGEST message/round accounting used here. These results target worst-case round complexity as the primary metric, in contrast to this paper's emphasis on simplicity of message structure: three message types and six modules.

2.3. Dynamic MST: Centralized Settings

A separate line maintains an MST under updates in a *centralized* model with full access to the graph in memory. Sleator and Tarjan's dynamic trees [28] underlie much of it (alongside Karger, Klein and Tarjan's randomized linear-time static construction [29]); Frederickson [30] gave the first $O(\sqrt{m})$-per-update structures; Eppstein, Galil, Italiano and Nissenzweig's sparsification [31] improved this generically; Holm, de Lichtenberg and Thorup [32] gave the first polylogarithmic fully-dynamic algorithm, and Eppstein et al. [33] handled the planar case. A related sub-problem (maintaining a replacement for every tree edge simultaneously) culminates in a recent engineered solution [16]. Recent engineering work continues the centralized line: Luo et al. [34] update an MST under batches of edge and vertex changes by reusing the surviving portion of the previous tree and continuing Kruskal's algorithm from it, motivated explicitly by backbone maintenance in ad hoc IoT networks—again presuming full in-memory access to the graph. This centralized line differs from ours in two ways worth making explicit. First, *what is maintained*: the replacement-maintenance approach [16] eagerly maintains a replacement for every tree edge (affordable on a single machine) whereas our ExactSwap (Section 5) computes a replacement reactively. Second, and more fundamentally, *what "local information" means*: a centralized algorithm can determine the true global minimum crossing edge directly, whereas a distributed algorithm cannot in general certify that a partial search has found the true minimum without examining a provably sufficient region (Section 5.5, Proposition 4)—an obstruction with no analogue in the centralized setting.

2.4. Dynamic MST: Sensor Networks and Distributed Settings

Awerbuch, Cidon and Kutten [35] studied optimal maintenance of a (not necessarily minimum) spanning tree under topology change, establishing the classical distributed-maintenance baseline; the present paper maintains the *minimum* spanning tree specifically. Díaz et al. [15] proposed a distributed dynamic GHS (DGHS) for wireless sensor networks, motivated by GHS's lack of tolerance to change: node failures and time-varying channel quality require the tree to be repaired rather than rebuilt, since rebuilding costs sensor nodes scarce battery energy. This motivation is essentially our own for Section 5, applied to a different base algorithm; DGHS does not address the CONGEST complexity questions our Claims 4–6 resolve, nor does it include a learning-augmented layer. A further distributed-maintenance line is the self-stabilizing one: Gupta and Srimani's self-stabilizing multicast protocols for ad hoc networks [36] maintain a spanning structure that recovers from arbitrary transient faults, an objective adjacent to but stronger than ours. Two lines address exactly this setting—a fixed communication topology whose edge weights change. Peleg [37] gave a distributed correction algorithm for minimum-weight spanning trees after weight changes, which stores the full tree at each node. Foerster et al. [38] formalize the batch dynamic CONGEST model, in which a batch of α weight changes arrives at once, and maintain an MST in $O(\alpha + d)$ rounds with only $O(\log n)$ bits of auxiliary state per node, together with an $\Omega(\alpha/\log^2 \alpha + d)$ lower bound that is tight up to a polylogarithmic factor in α. Their accounting is per batch, in rounds and space, and their algorithm broadcasts the changed labels network-wide to compute a matroid basis; ours is per single update, in messages, and confines work to the affected region, whose size we measure directly (Section 7). The two are complementary: for a single change their round bound is $O(d)$, comparable to ours, but the message cost of a global broadcast is incurred regardless of how localized the actual repair is.

2.5. Learning-Augmented Algorithms

Lykouris and Vassilvitskii [17] introduced the modern framework; Purohit, Svitkina and Kumar [39] established its consistency–robustness tradeoffs for scheduling problems, and Mitzenmacher and Vassilvitskii [18] survey the area. Van den Brand et al. [19] extend it to dynamic graph algorithms (reachability, shortest paths, matching) in a centralized setting, with predictions concerning the timing of future updates. Their work addresses neither MST nor the distributed CONGEST model. We investigate whether their framework extends to distributed MST; the answer is instructive rather than straightforwardly positive—Section 5.5 shows that the partial-search verification their mechanisms would require has no sound distributed analogue for this problem. Section 6 accordingly repositions the learning-augmented contribution around scheduling decisions that do not require certifying a shortcut's correctness.

2.6. Adjacent Distributed P2P and Overlay Tree Constructions

The most directly comparable work is that of Abdel-Wahab et al. [40]: a fully distributed MST algorithm for wide-area communication networks, motivated, like ours, by efficient multicast. It grows disjoint trees concurrently, each repeatedly joining its closest neighbouring tree, until a single tree remains; the construction is fragment-growing in the GHS lineage and addresses the static problem only. The author's prior work [13] applies the NTE static construction within δ, μ-bounded MON-Clusters for P2P video streaming. Hossain and Khan's ZePoP [41] elects a delay-optimal leader by closeness centrality and builds a data collection and distribution tree rooted at it, optimizing delay rather than edge-weight minimality; a centrality-based construction and an MST-based one solve genuinely different problems, and their outputs need not coincide. Mokhtarian and Jacobsen [42] address minimum-delay overlay multicast, a distinct objective. At the infrastructure level, structure-placement and lifetime-optimization problems—load-balanced gateway placement in wireless mesh backbones [43], lifetime optimization of dense sensor deployments [2], and aggregation-point placement for semi-decentralized federated learning [44]—optimize related objectives at coarser granularity; the present work supplies the tree-*maintenance* primitive that such infrastructure-level optimizers implicitly assume.

3. Model and Preliminaries

3.1. Graph Model

Let $G = (V, E, \omega)$ be a connected, undirected graph with $|V| = n$ nodes and $|E| = m$ edges, where $\omega : E \to \mathbb{R}^+$ assigns a positive weight to each edge. We write d for the diameter and Δ for the maximum degree. As is standard in CONGEST, edge weights are assumed polynomial in n, so a weight fits in $O(\log n)$ bits.

Assumption 1 (Unique weights)**.** *All edge weights are distinct. This is without loss of generality: ties break by the pair* $(\omega(e), \mathrm{id}(e))$*, ordered lexicographically:* $e \prec e'$ *iff* $\omega(e) < \omega(e')$*, or* $\omega(e) = \omega(e')$ *and* $\mathrm{id}(e) < \mathrm{id}(e')$*. This total order is used wherever edge weights are compared—in the orientation rule (Section 4.3) and in every candidate comparison of the detection and maintenance phases. Uniqueness makes the MST unique. In practice, quantized or coarsely reported link metrics produce many exact ties; the identifier tie-break then decides deterministically, correctness is unaffected, and the only consequence of near-equal weights is that a different—equally valid—minimum spanning tree may be selected.*

Definition 1 (MST; tree and non-tree edges)**.** *The minimum spanning tree* $\mathrm{MST}(G)$ *is the acyclic edge subset* $T \subseteq E$ *connecting all of* V *with minimum total weight. Edges in* T *are* tree edges*, and edges in* $E \setminus T$ *are* non-tree edges *(NTEs). For an NTE* (u, v)*, its* fundamental cycle

is the cycle formed by (u, v) *and the unique tree path between* u *and* v*; the tree edges on that path are the edges the cycle* crosses.

3.2. Distributed Computation Model

We work in the synchronous CONGEST model [22]. A processor resides at each node; computation proceeds in synchronous rounds; each round, every node sends one message of at most $O(\log n)$ bits across each incident edge. Links are bidirectional and reliable. Each node initially knows its own $O(\log n)$-bit identifier, its neighbors' identifiers, and its incident edge weights. Cost is measured in total messages (one message across one link = one unit) and rounds. At least one node, the *session initiator*, is awake at the start and designates an *update initiator*.

3.3. Slate State

Each node maintains local state called its *slate*, specified in Section 4. Two fields on which the dynamic extension depends on:

Definition 2 (Subtree size)**.** *With the tree rooted per the orientation of Section 4,* $sz(v)$ *is the number of nodes in the subtree rooted at* v*, maintained as a byproduct of reply-aggregation at no additional asymptotic cost.*

Definition 3 (Candidate weight)**.** *For each NTE* e *recorded in a node's slate, the field* $candidate_weight[e]$ *stores the maximum tree-edge weight on* e*'s fundamental cycle as of the last refresh. The field is populated by a post-construction labeling pass (Section 4)—not from the comparison value observed at detection time, which provably differs from the fundamental-cycle maximum in general (Remark 1)—and thereafter maintained by affected-set refresh. It is what makes the* $O(1)$ *stability check of Lemma 1 possible.*

3.4. Dynamic Update Model

Definition 4 (Update sequence)**.** *An update sequence is* $\sigma = \langle u_1, \ldots, u_T \rangle$*, where each* $u_t = (e_t, \omega'_t)$ *changes the weight of one existing edge* $e_t \in E$ *to* ω'_t*, preserving uniqueness (Assumption 1). Edge insertions and deletions, and hence node arrival and departure, are outside this model and are deferred to future work; Section 5.4 shows how each case reduces to the machinery developed here. Updates are observed initially only by the endpoints of* e_t*, and are well-separated in time. Simultaneous or bursty changes are handled by serialization: concurrent updates enter a queue and are processed one session at a time, which is exactly the contended regime evaluated in Section 7 (a stream of 4000 updates under a fixed per-round budget), where scheduling order becomes the performance lever.*

Real link costs exhibit two kinds of variation on very different timescales, a distinction long established by Internet measurement studies. The first is short-timescale jitter: delay fluctuation from queuing and contention, small relative to the weight gaps between distinct edges. Such fluctuation does not change the relative order of edge weights, and therefore cannot change the MST, by Assumption 1 and Path Optimality (Theorem 2). Measurement studies support this: path properties such as round-trip time are operationally steady over the intervals that matter here [45]. Discrete structural events—route changes, point-of-presence reassignment, link re-provisioning—occur on the scale of minutes to weeks [46] and *do* change the optimal tree. Only the latter are modeled as elements of σ; node failure, which removes a vertex rather than re-weighting an edge, is the topology-change case deferred above: a practical implementation applies a hysteresis threshold so that jitter never triggers processing, and the well-separated-updates assumption is realistic under this reading, since the per-update processing cost of Section 5 is far below typical inter-

arrival times of structurally meaningful changes. The same two-timescale structure holds in wireless sensor networks, where link quality drifts with interference and mobility while re-routing and transmission-power adjustment shift link costs abruptly; node death from battery depletion is again the deferred topology-change case. After each u_t is processed, the invariant $T_t = \mathrm{MST}(G_t)$ holds, with each node's slate current.

4. The Static NTE Algorithm

This section gives the full specification, correctness proofs, and complexity of the NTE-tracking algorithm, introduced in outline in [13] and originating in [20]. The conference presentation established the two-phase structure, the module decomposition, and proofs of Path Optimality and graph orientation, but stated complexity without proof; we supply the full module-level specification and slate design, the Cycle Theorem, Theorem NTE, and the complexity proofs.

4.1. Overview

The algorithm's central design choice is to identify the non-tree edges—the set whose membership has a strictly local certificate (Path Optimality, Theorem 2)—rather than growing tree fragments. It runs in two phases: an *orientation phase* converting the undirected graph into an acyclic directed graph via *request* and *reply* messages, and an *NTE detection phase* in which *update* messages flow from the update initiator back toward the session initiator, each NTE identified by exactly one node en route. Six modules generate and aggregate the three message types.

4.2. Messages and Slate Variables

Each node's slate is initialized with $level_j = 0$, the sentinel value meaning *unleveled*; levels are assigned by the request flood of the orientation phase (Section 4.3), empty parent, child, and NTE sets, and zeroed aggregation fields. Three message types are used. A *request* carries six fields (sender/receiver id, sender level, session-initiator id, update-initiator id, session id); a *reply* carries three (sender/receiver id, ACK/NACK); an *update* carries six (sender/receiver id, candidate-edge, max-link, non-shared-candidate-edge, max-link-non-shared-path). All fields are $O(\log n)$ bits (Claim 1).

Each node's slate holds its id, neighbor set, incident edge costs, level, parent/child sets, the request sets rr, rs, and the working fields *smallest-max-link*, *candidate-edge*, the detected-NTE set, and *candidate_weight*$[\cdot]$ (Definition 3).

Remark 1 (The aggregation runner-up is not the fundamental-cycle maximum)**.** *At the moment a node detects an NTE by comparing two branch max-links, the runner-up value is the maximum weight along the* alternative aggregation path back to the update initiator*—which may traverse non-tree edges and omit tree edges of the NTE's fundamental cycle. In Figure 1 graph, simulation shows that the runner-up diverges from the fundamental-cycle maximum on four of the six NTEs, in both directions. Divergence either way is unsafe for the $O(1)$ check of Lemma 1: a too-high value triggers spurious swaps, a too-low one silently misses required swaps. The runner-up therefore must not be used as candidate_weight.*

The field is instead populated by a post-construction labeling pass over the final tree, assigning each node a label supporting maximum-edge-weight queries on tree paths (in the Korman–Kutten line of distributed MST-verification labeling [47], whose label size $O(\log n \log W)$ is optimal, with W the largest edge weight; under the polynomial-weight convention of Section 3 this is $O(\log^2 n)$ bits), after which each NTE's endpoints compute the fundamental-cycle maximum from their labels. The pass is required only for the dynamic extension; a static-only deployment may omit it.

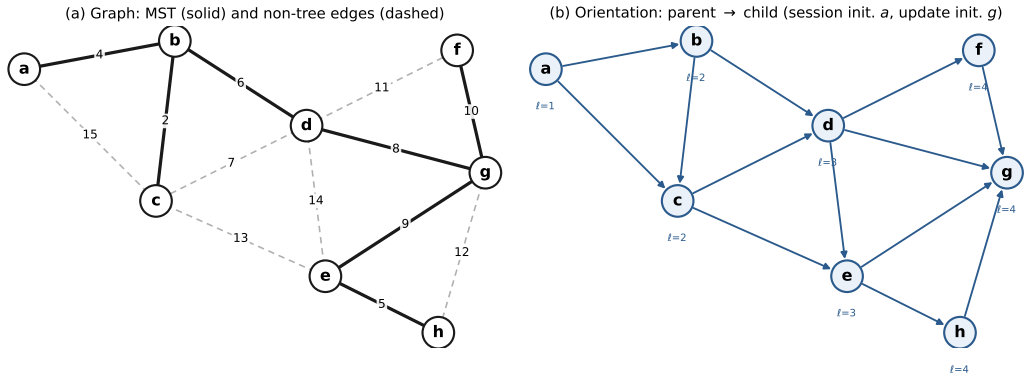

Figure 1. The worked-example graph of Section 4.5. (**a**) The weighted graph with MST edges solid and non-tree edges dashed. (**b**) The oriented DAG produced by the orientation phase, with levels ℓ and parent→child arcs; session initiator a, update initiator g.

4.3. Graph Orientation

For node v, let $\mathrm{id}(v)$ be its identifier, $\mathrm{nbr}(v)$ its neighbors, and $rr(v)$ the nodes from which it received a request. The level function $f : V \to \{1, \ldots, n\}$ assigns each node the position at which it first received a request; we write $l_v = 0$ as a sentinel value denoting that v has not yet been leveled:

$$l_v = \begin{cases} 1 & v = \text{session initiator} \\ l_x + 1 & x \in rr(v) \text{ first sender, } f(x) = l_x \\ 0 \text{ (unleveled)} & rr(v) = \emptyset,\ v \neq \text{session initiator} \end{cases} \tag{1}$$

The third case is transient: it describes a node the request wave has not yet reached, and it is precisely the state tested by the request aggregator's opening guard (Section 4.4), which enforces the first-request rule. Since G is connected and requests flood to all neighbors, every node is leveled within $O(d)$ rounds (see Claim 3), so f is total over V at termination and the sentinel never survives to the end of the orientation phase. The slate initializes $level_j = 0$ (Section 4.2), matching this sentinel convention. Each edge (v_i, v_j) is oriented so that v_i is the *parent* of v_j when

$$(v_i \to v_j) \quad \text{if} \quad l_i < l_j \ \text{ or } \ (l_i = l_j \wedge \mathrm{id}_i < \mathrm{id}_j) \\ \text{or } \ v_j = \text{update initiator}. \tag{2}$$

Levels are assigned once per update session by the request flood and remain fixed for the remainder of that session; the orientation of Equation (2) is therefore static throughout the detection phase, and changes only when a new session re-floods.

Orienting the edges of a graph by a vertex numbering, so that every edge points from lower to higher number, is a standard way to obtain an acyclic orientation [48]. The rule below instantiates this idea with a numbering the protocol can compute in one flood—levels first, identifiers as tie-breakers—plus an override that makes the update initiator the unique sink; the statement records the properties this specific rule guarantees.

Theorem 1 (Graph Orientation)**.** *A directed graph is acyclic if and only if its nodes can be numbered such that, for every arc* (v_i, v_j), $l_i \leq l_j$.

Proof. ($\Rightarrow$) Suppose a cycle exists on v_i, v_j, v_k with arcs (v_i, v_j), (v_j, v_k), (v_k, v_i). If $l_i \neq l_j \neq l_k$, arcs force $l_i < l_j < l_k$, so $l_k > l_i$, contradicting (v_k, v_i), which, by (2) requires $l_k \leq l_i$. If

$l_i = l_j = l_k$, the tie-break forces $\mathrm{id}_i < \mathrm{id}_j < \mathrm{id}_k$, again contradicting closure. If $l_i = l_j \neq l_k$, then $l_i < l_k$ contradicts (v_k, v_i). ($\Leftarrow$) Symmetric case analysis on level/id ordering. □

Orientation guarantees that the detection phase propagates without deadlock: every message flows from a lower level to a higher one. Levels are assigned by breadth-first arrival, so a node adjacent to two differently-leveled nodes takes the smaller-level parent's level plus one.

4.4. The Six Modules

The algorithm uses six event-driven modules: request generator, request aggregator/propagator, reply aggregator, update generator, update aggregator, and the degenerate-case handler. Pseudocode is given in Appendix A; the update aggregator, the heart of the detection phase, appears as Algorithm 1.

Algorithm 1 Update Aggregator (at internal node s)

1: **on** receiving updates from every child $c \in \mathrm{child}(s)$:
2: **if** exactly one child **then**
3: $\quad$ *smallest-max-link* $\leftarrow$ $maxlink_c$; *candidate-edge* $\leftarrow$ $cand_c$
4: **else**
5: $\quad$ sort children by *maxlink* descending as $c_1, c_2, \ldots$
6: $\quad$ detect NTE $cand_{c_1}$ ▷ Proposition 2
7: $\quad$ **if** $maxlink_{c_1} = maxlink_{c_2}$ **then**
8: $\quad\quad$ resolve via non-shared-path comparison
9: $\quad$ **end if**
10: $\quad$ *smallest-max-link* $\leftarrow$ $maxlink_{c_2}$; *candidate-edge* $\leftarrow$ $cand_{c_2}$
11: **end if**
12: **for** each $p \in \mathrm{parent}(s)$ **do**
13: $\quad$ send UPDATE to p with max-link $\leftarrow \max(\textit{smallest-max-link}, \mathrm{cost}(s, p))$
14: **end for**

The aggregation step, in words (Figure 2). The update phase is a wave that starts at the update initiator D (the unique childless node of the orientation) and flows child-to-parent toward the session initiator. Every UPDATE message carries two values: *max-link*, the largest edge cost on the path the wave has traversed from D to the sender, and *candidate-edge*, the identity of the edge attaining it. A node with a single child merely extends the path: it forwards the pair, raising max-link if its own outgoing edge is heavier (line 13). Detection happens where the wave *reconverges*: when a node s has received from two (or more) children, the two branch paths from D to s together close a cycle—this is the situation of Proposition 1—and the heaviest edge on that cycle is simply the larger of the two branch max-links (line 6). By Proposition 2 that edge cannot belong to the MST, so s records it as an NTE (line 7). The *smaller* max-link—the runner-up—describes the surviving path from D to s and is what continues upward (line 12), again raised by $\mathrm{cost}(s, p)$ as the wave proceeds (line 14). Figure 2 traces one such meeting; ties between branch maxima are broken by the non-shared-path comparison of lines 8–10, which consults only the portions of the two paths beyond their last common node.

4.5. Worked Example

Consider Figure 1 with weights $ab = 4$, $ac = 15$, $bd = 6$, $bc = 2$, $de = 14$, $dc = 7$, $df = 11$, $dg = 8$, $eg = 9$, $ec = 13$, $eh = 5$, $fg = 10$, $gh = 12$; session initiator a, update initiator g. Independent recomputation gives tree edges $\{ab, bc, bd, dg, eg, eh, fg\}$ (weight 44) and NTEs $\{ac, cd, ce, de, df, gh\}$. A full simulation of the oriented-DAG aggregation reproduces exactly six detections—gh at e, ed and fd at d, ec at c, dc at b, ca at a—matching the independently computed NTE set.

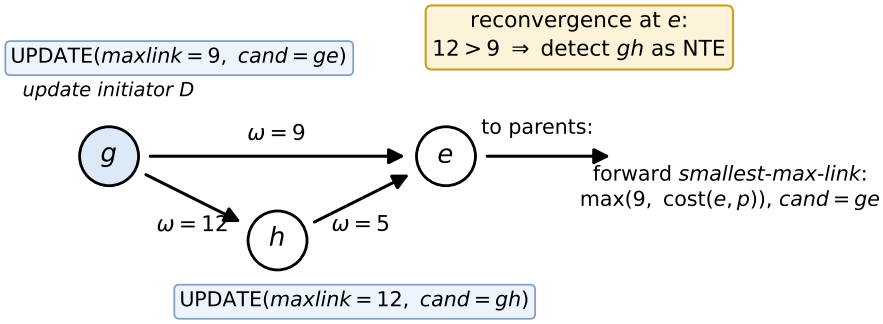

Figure 2. The aggregation step on the worked example of Figure 1 (update initiator $D = g$). Two branch paths reconverge at e: the child g reports $(maxlink = 9,\ cand = ge)$ and the child h reports $(12, gh)$. The larger max-link identifies its candidate edge gh as a non-tree edge (Proposition 2); the runner-up pair (the pseudocode's *smallest-max-link*) is forwarded to e's parents, raised by the outgoing edge cost (Algorithm 1, line 13). Exactly one cycle is closed per reconvergence, which is the content of Proposition 1.

4.6. Correctness

The following characterization is classical: it is equivalent to the cut and cycle properties of minimum spanning trees [48,49] (see also the Path Optimality conditions of network-flow theory). We state it in the form the protocol exploits and include the one-paragraph proof for self-containment; a proof also appears in the conference paper [13].

Theorem 2 (Path Optimality)**.** *A spanning tree T^* is an MST if and only if, for every non-tree edge (k,l), $\omega(i,j) \leq \omega(k,l)$ for every edge (i,j) on the T^*-path between k and l.*

Proof. ($\Rightarrow$) If (i,j) on the path had $\omega(i,j) > \omega(k,l)$, swapping (i,j) for (k,l) gives a lighter spanning tree, contradicting minimality. ($\Leftarrow$) If the inequality holds throughout, no single-edge swap decreases total weight, and since any spanning tree reaches any other by such swaps, T^* is minimum. □

Proposition 1 (Cycle Theorem)**.** *If two paths converge at a node during the detection phase, exactly one cycle—and hence exactly one non-tree edge—is produced.*

Proof. Let aggregation occur at s with update initiator u. Paths p_i, p_j share an initial segment to some k, then diverge through distinct $q \neq r$ (distinct because if $q = r$, node q would already have detected the cycle and forwarded one path), reconverging at s. Then $p_i \cup p_j$ is a single cycle, introducing exactly one chord. □

Proposition 2. *The edge of maximum cost among the two paths aggregated at a node is a non-tree edge.*

Proof. Immediate from Theorem 2 and Proposition 1: the heaviest edge on the unique cycle cannot lie in any MST containing the cycle's other edges. □

Theorem 3 (Theorem NTE)**.** *Let NTE_i be the NTEs detected at node i. Then $G - \bigcup_i NTE_i = \mathrm{MST}(G)$.*

Proof. Via three lemmas. *(i) $G - \bigcup NTE_i$ is a spanning tree.* Every node retains exactly one surviving path to the update initiator D (the update aggregator keeps only the smaller-max-link branch), so the result is connected, spans V, and is acyclic. *(ii) Each detected kl is genuinely a non-tree edge of G.* Detection at v via candidate ij implies $\omega(ij) < \omega(kl)$ (Proposition 2); any alternative k–l path through another node u has all edges lighter still,

so by Path Optimality kl remains non-tree. *(iii) No NTE remains undetected.* By (i) the result is already a tree with $n-1$ edges; removing more would disconnect it. Combining (i)–(iii) with Theorem 2 and Proposition 1 gives the claim. □

4.7. Complexity

Claim 1 (Message size)**.** *Every message is $O(\log n)$ bits.*

Proof. Identifiers are $O(\log n)$ bits; the level is bounded by n. Each message carries a constant number of such fields. □

Claim 2 (Message complexity)**.** *The algorithm uses $O(m)$ messages.*

Proof. At most $2m$ requests (one per incident edge per side, plus at most one conflict resolution), $2m$ replies, and m updates (one per oriented parent-child pair): at most $5m$ total. □

Claim 3 (Time complexity)**.** *The algorithm completes in $O(d)$ rounds.*

Proof. Request propagation from session to update initiator is $O(d)$; replies proceed concurrently; update propagation back is $O(d)$. □

Table 1 compares against prior algorithms. The NTE advantage is not asymptotically better message complexity than the best modern CONGEST results, but a smaller message-type count and simpler per-message structure—properties that matter for the dynamic extension, where each message type's role in affected-set maintenance must be tracked explicitly. When the dynamic extension is deployed, the labeling pass adds its own cost (a distributed maximum-weight-on-tree-path scheme in the Korman–Kutten line [47]) as a separate line item; a static-only deployment does not incur it.

Table 1. Comparison with prior distributed MST algorithms. Here d is the network diameter (Section 3); $\tilde{O}$ hides polylogarithmic factors. † Randomized (Las Vegas); all other entries, including ours, are deterministic. GKP analyze time only and do not bound message complexity.

Algorithm	Messages	Time	Msg Size	Msg Types
GHS [7]	$O(n \log n + m)$	$O(n \log n)$	$O(\log n)$	∼5
GKP [9]	not analyzed	$O(d + n^{0.614})$	$O(\log n)$	several
LPP [11]	—	$O(\log \log n)$	—	overlay
PRS [12] †	$\tilde{O}(m)$	$\tilde{O}(d + \sqrt{n})$	$O(\log n)$	several
NTE (this paper)	$\mathbf{O(m)}$	$\mathbf{O(d)}$	$\mathbf{O(\log n)}$	**3**

5. Dynamic NTE-MST Framework

This section presents a correct, complete, non-learned algorithm for maintaining the MST under edge-weight updates, built on Section 4. Section 6 builds a scheduling layer on top; nothing there is required for this section's claims.

5.1. Case Analysis

An update (e, ω') falls into exactly one of four cases:

- **Case A** (non-tree edge, weight increases): e remains non-tree by Path Optimality. Cost $O(1)$, no messages.
- **Case B** (non-tree edge, weight decreases): decided in $O(1)$ by Lemma 1; if a swap results, the leaving edge is known from the slate—no search.
- **Case C** (tree edge, weight decreases): e remains a tree edge. Cost $O(1)$.

- **Case D** (tree edge, weight increases): the one case that can force a structural change; resolved by ExactSwap (Section 5.2).

Lemma 1 (Stability Verification, Case B)**.** *Let (k, l) be a non-tree edge, and let w_c denote its stored candidate_weight value: the maximum tree-edge weight on its fundamental cycle (Definition 3). On update ω': if $\omega' \geq w_c$, then (k, l) remains non-tree, verifiable in $O(1)$ by one local comparison; if $\omega' < w_c$, then (k, l) enters the tree and the edge achieving w_c leaves it—a swap whose leaving edge is already identified by the endpoint labels (Section 4.2), so that, in contrast to Case D,* no search is required*: the splice and the slate refresh reuse* ExactSwap*'s update machinery while skipping its search phase entirely.*

Proof. Immediate from Path Optimality (Theorem 2): (k, l) stays non-tree exactly when its weight is not less than the heaviest tree edge on its fundamental cycle, which is w_c. The comparison needs no messages since w_c is local slate state. □

Case B can therefore change the tree's edge set; what distinguishes it from Case D is not the absence of structural change but the absence of *discovery*. In Case B the entering edge is the updated edge itself and the leaving edge is determined by the state the endpoints already hold; in Case D the replacement must be found, and Proposition 4 shows that finding it provably requires searching the full region. "Three of four cases resolve in $O(1)$" refers to this decision cost.

5.2. The Affected Set and the ExactSwap Protocol

Let Case D occur at tree edge $e = (p, c)$, p the parent of c. Removing e splits the tree into T_c (rooted at c) and $\bar{T}_c = V \setminus T_c$.

Definition 5 (Affected set)**.** *The affected set of a Case-D update at e is the set of non-tree edges whose fundamental cycle crosses e—equivalently, those with exactly one endpoint in T_c. These are exactly the NTEs whose candidate_weight becomes stale when e leaves the tree.*

The protocol ExactSwap: (1) since p, c both know $sz(c)$, let T_{small} be the smaller of $T_c, \bar{T}_c$ ($|T_{\text{small}}| \leq n/2$); (2) the root of T_{small} floods a one-bit marker to its members; (3) each marked node probes its *non-tree* incident edges for cut-crossing candidates; (4) a convergecast reports the minimum-weight crossing edge; (5) if lighter than $\omega'(e)$, it replaces e, and affected slates update by piggyback on steps 3–4.

Step 3's restriction to non-tree edges rests on a structural fact: within a spanning tree, every edge crossing the cut induced by removing e, other than e itself, is necessarily a non-tree edge—so tree-edge neighbors need not be probed.

5.3. Complexity of ExactSwap

Claim 4 (Messages)**.** ExactSwap *resolves a Case-D update in $O(\sum_v \delta_{\text{NTE}}(v))$ messages, the sum over $v \in T_{\text{small}}$, where $\delta_{\text{NTE}}(v)$ is v's non-tree degree.*

Proof. Steps 2 and 4 cost $O(|T_{\text{small}}|)$ each; step 3, by the structural fact above, probes only non-tree incident edges, costing $O(\sum_{v \in T_{\text{small}}} \delta_{\text{NTE}}(v))$, which dominates. □

Claim 5 (Rounds)**.** ExactSwap *resolves a Case-D update in $O(\text{depth}(T_{\text{small}}))$ rounds, which is at most $O(d)$.*

Claim 6 (Message size)**.** *All* ExactSwap *messages are $O(\log n)$ bits.*

The message bound of Claim 4 is empirically 9–42% of the naive degree-based bound $O(\Delta \cdot \min(|T_c|, n - |T_c|))$, with the largest improvement on scale-free graphs (Section 7).

5.4. Beyond Weight Updates: Insertions, Deletions, Departures

The update model of Section 3 covers weight changes to existing edges; the machinery above nonetheless already decides the natural extensions. For *insertion*:

Proposition 3 (Insertion decision)**.** *Let a new edge (k, l) with weight ω be added between existing nodes, and let w_c be the maximum tree-edge weight on the k–l tree path, available from the endpoint labels (Section 4.2). If $\omega > w_c$, the tree is unchanged; if $\omega < w_c$, (k, l) enters and the edge achieving w_c leaves. The decision is one local comparison.*

Proof. Path Optimality (Theorem 2) applied to the single fundamental cycle the new edge closes. □

Deletion of a tree edge reduces to Case-D machinery with the removed edge unavailable as its own replacement, and node departures compose multiple deletions. What a full treatment adds beyond these reductions is slate initialization and teardown for arriving and departing nodes under realistic churn; that is deferred to follow-up work (Section 9).

5.5. On Partial-Search Shortcuts: A Distributed–Centralized Separation

Proposition 4 (No sound local certificate for partial-search verification)**.** *There is no distance-based stopping rule that certifies, using only information visible within a bounded region around c, that the true minimum-weight edge crossing the cut induced by e has been found, for arbitrary edge-weight assignments.*

Proof. By construction. Consider a tree path from c with a crossing edge of weight w_1 at distance r_1 and a lighter crossing edge of weight $w_2 < w_1$ at greater distance $r_2 > r_1$, with no crossing edges in between. A search halting upon finding the weight-w_1 edge and observing a quiet local frontier cannot detect the lighter edge further out, since edge weight is assigned independently of tree distance from c: any stopping rule based on the local appearance of the frontier is defeated by placing the true minimum arbitrarily far. □

Consequently, verifying the true minimum crossing edge with a worst-case guarantee requires examining the entire relevant region—precisely Claim 4's cost, not a looseness in that claim. This is a point of contrast with the centralized dynamic-MST literature (Section 2), where global visibility makes the question moot. An eager-caching alternative (maintaining a replacement per tree edge) was investigated; it handles one invalidation trigger cleanly but a second (a Case-B decrease silently undercutting an existing cache entry) remains unresolved, so it is recorded as future work rather than adopted.

5.6. Summary: The Non-Learned Floor

Combining the above, the non-learned dynamic algorithm maintains $T_t = \mathrm{MST}(G_t)$ after every update at cost $O(1)$ for Cases A and C, and for Case B without a swap, and $O(\sum_{v \in T_{\text{small}}} \delta_{\text{NTE}}(v))$ messages / $O(d)$ rounds for Case D or a Case-B-triggered swap—always at most $O(m)$, and by Proposition 4, the true cost with no sound distributed shortcut.

6. Learning-Augmented Scheduling

Since the expensive computation cannot be shortcut (Proposition 4), prediction is used to decide *when* it runs, not to reduce its cost. On a Case-D event, a predictor estimates whether the event is cheap or expensive and, for bounded-staleness-tolerant applications,

expensive events may be deferred to a maintenance window—the application meanwhile continuing on the last-known-correct tree—while cheap updates are applied immediately.

Proposition 5 (No correctness cost from the predictor)**.** *The scheduling-aware algorithm maintains $T_t = \mathrm{MST}(G_t)$ at every point where a deferred update is resolved, identically to the non-learned algorithm, regardless of predictor accuracy.*

Proof. Every branch that resolves a Case-D event does so via unmodified ExactSwap (Claim 4); the predictor only selects which branch is taken when, never altering what any branch computes. □

The value of scheduling, namely improved throughput or responsiveness under contention, depends on workload and is properly an empirical question (Section 7). The predictor uses local graph-structural features already available in the slate: node degrees, non-tree degree, $sz(c)$, and edge-weight ranks. Notably, on scale-free topologies the dominant feature is $sz(c)$ itself, so a threshold on existing slate state largely suffices, and where a model does help (denser topologies), a *linear* one recovers nearly all of the signal (Section 7)—so the "learning" here is deliberately lightweight. This is a more modest role than a cost-reduction guarantee in the style of van den Brand et al. [19]; resolving the open eager-caching question (Section 5.5) could reopen a stronger result in future work.

7. Experimental Evaluation

Our evaluation is designed around the question an infrastructure operator would ask: how does maintenance cost behave *at scale*? We therefore measure across graph sizes up to $n = 20{,}000$ (221,874 update records), report how per-update cost scales with deployment size, and give particular weight to the scaling of the affected set (Section 7.2), since it determines whether the approach remains viable as an infrastructure grows.

All CONGEST costs are computed analytically from the proven formulas (Claims 1–3, 4–6) applied to NetworkX-generated graphs; step-by-step simulation is used only for the GHS baseline, which lacks a closed-form cost. Graphs: Barabási–Albert (BA, $m = 2$ attachment—sparse, scale-free, closest to our applications), Erdős–Rényi (ER), and random geometric (GEO). All runs are seeded and reproducible.

7.1. Static Construction: NTE vs. GHS

We simulate the standard synchronous phase-based GHS (distributed Borůvka with GHS-style test/reject pruning), with per-phase message and depth-based round accounting; every simulated run's output was verified against an independently computed MST. GHS is measured; NTE is charged its proven worst case ($5m$ messages, $\sim 3d$ rounds)—an asymmetry favoring GHS. On rounds, NTE wins everywhere (GHS 1.3–1.75× more, across all families and sizes). On messages, a density-dependent crossover: on sparse BA, GHS uses ∼2.2× more; on GEO, ∼1.5× more; on dense ER, GHS uses fewer (ratio 0.65–0.84), because its dominant per-edge cost is the one-time two-message test/reject plus $O(n \log n)$ overhead while NTE's bound scales as $5m$. NTE's message advantage is thus a sparse-graph advantage—and our motivating applications are sparse by design. The crossover can be located exactly on a guarantee-versus-guarantee basis: GHS's classical bound is $2m + 5n \log_2 n$ messages [7], so NTE's worst case of $5m$ is smaller precisely when the average degree is below $\frac{10}{3} \log_2 n$—a threshold of 22 at $n = 100$, 33 at $n = 1000$, and 44 at $n = 10{,}000$, growing with scale. *Every configuration in Table 2 sits below this threshold* (our densest, ER at $n = 3000$, has average degree 17.6 against a threshold of 38.5): on a like-for-like worst-case basis NTE wins in every tested regime, and the measured reversal on ER arises from the deliberately asymmetric charging—measured GHS runs at 38–48% of its

bound on these graphs while NTE is billed 100% of $5m$. Two further observations bound the objection's reach. First, the *round* advantage persists at every density measured (1.45–1.75× on ER). Second, the maintenance framework of Sections 5 and 6 is construction-agnostic: its inputs—incident tree/non-tree status, parent/child orientation, subtree sizes (one tree convergecast), and the *candidate_weight* labels of the post-construction labeling pass—are computable from *any* rooted MST, so a genuinely dense deployment may construct with GHS or a modern optimal algorithm [12] and still obtain every maintenance guarantee of this paper; the crossover concerns the one-time build, not the per-update economics that are our central subject. Table 2 gives the measured comparison.

Table 2. Static construction: measured synchronous GHS versus NTE charged its proven worst case ($5m$ messages, $\sim 3d$ rounds). Means over 3–10 seeded graphs per row; ratios are GHS ÷ NTE.

Fam.	n	m	GHS msg	GHS rnd	NTE msg	NTE rnd	msg Ratio	rnd Ratio
BA	100	196	1794	74	980	56	1.83	1.32
BA	1000	1996	22,355	176	9980	118	2.24	1.49
BA	3000	5996	66,114	206	29,980	130	2.21	1.59
ER	100	496	2088	110	2482	76	0.84	1.45
ER	1000	7597	28,530	393	37,987	225	0.75	1.75
ER	3000	26,383	85,742	564	131,915	356	0.65	1.58
GEO	1000	5557	42,425	605	27,788	374	1.53	1.62
GEO	3000	19,497	145,659	1212	97,487	776	1.49	1.56

7.2. Case Distribution and Sweep-Size at Scale

Across 221,874 tree-edge records (BA/ER at $n = 500$–5000 and GEO at $n = 500$–2000 in the full three-family sweep, extended on BA to $n = 10{,}000$ and 20,000): the median affected-set size on BA is exactly 2 at every scale, with 67% of Case-D events touching ≤ 3 NTEs—a scale-invariance strengthening the small-scale finding. ER's median grows ($25 \to 35$) and mean grows steeply ($116 \to 505$); GEO grows moderately. Claim 4's refined bound correlates with the true affected-set size at $r = 0.985$ (BA), 0.984 (ER), 0.746 (GEO), with median ratio 1.00—the bound is essentially tight in practice, and exactly tight for 96% of events on BA, 83% on ER, and 55% on GEO. At the two largest scales the bound tightens further: exactly tight for 98% of events at both $n = 10{,}000$ and $n = 20{,}000$, with zero violations among the 89,994 extension records. In Figure 3b, every event lies on or above the diagonal (the bound is never violated) and points above it are events where locally-known non-tree degrees include edges internal to the smaller side, which the bound must count but the true affected set excludes: a node's slate knows its non-tree degree for free, but distinguishing cut-crossing from internal edges would require non-local information. This looseness is accordingly most common on GEO, where spatial locality keeps many non-tree edges within one side. Figure 3a's curve shapes also preview the scheduling result: BA's cliff-and-tail distribution (most events cheap, cost concentrated in a rare tail) is exactly the cheap/expensive separability that Section 7's PRIORITY policy exploits, whereas ER's uniformly high curve (92% of events exceed size 10) leaves a scheduler the least to separate. Table 3 details the affected-set distribution.

7.3. Magnitude Predictor: How Much Model Is Needed?

The scheduler (Section 6) needs only to distinguish cheap from expensive Case-D events. We ask how much modelling this requires, comparing four points on a complexity spectrum—a zero-parameter threshold, a linear model, and two tree ensembles—all evaluated with by-graph train/test splits (never by row), so every number is a generalization to unseen graphs. Features are local slate quantities (endpoint degrees, non-tree degrees, $sz(c)$, depth, edge-weight ranks, n, m); the label is "expensive" (refined bound > 10). Table 4 reports AUC and the Brier score (calibration; lower is better).

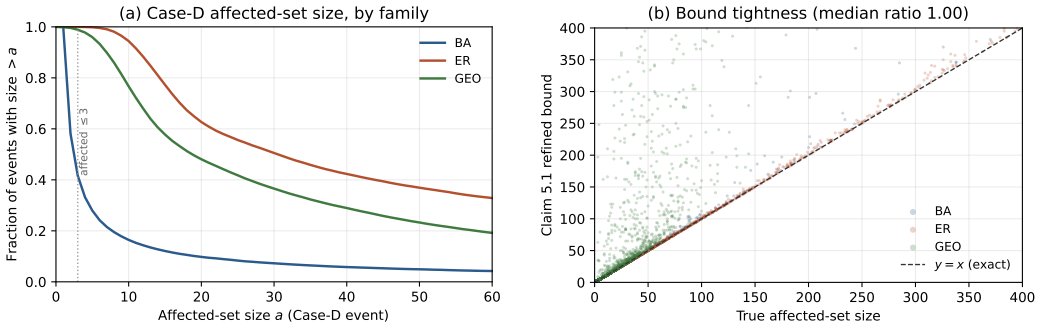

Figure 3. (**a**) Complementary CDF of Case-D affected-set size by family: BA is sharply concentrated (two-thirds of events touch ≤ 3 NTEs), while ER and GEO have heavy tails (axis truncated at 60; tails extend further—a third of ER events exceed it). (**b**) Claim 4's refined bound versus the true affected-set size; points cluster on $y = x$ (median ratio 1.00), no point lies below it (the bound is never violated), and points above it are conservative cases where the locally-counted non-tree degree includes edges internal to the smaller side.

Table 3. Affected-set size of Case-D events by family and scale. BA's median is scale-invariant at 2; ER and GEO grow with n. "≤ 3" is the fraction of events touching at most three NTEs. ‡ Scales added in revision: three graphs each (29,997 and 59,997 events), same generator and analysis.

Fam.	n	Median	Mean	p90	frac. ≤ 3
BA	500	2	10.5	18	0.67
BA	1000	2	13.1	19	0.66
BA	2000	2	15.8	19	0.67
BA	5000	2	19.4	19	0.67
BA	10,000 ‡	2	20.9	18	0.67
BA	20,000 ‡	2	26.1	18	0.67
ER	500	25	116.0	279	0.00
ER	5000	35	504.6	549	0.00
GEO	500	15	25.7	63	0.04
GEO	2000	22	53.9	150	0.01

Table 4. How much model is needed to predict Case-D cost. AUC (top) and Brier score (bottom), by-graph split, mean over 3 seeds. The threshold uses the slate's $\min(sz(c), n - sz(c))$ and emits a raw bound, not a probability (no Brier). A *linear* model already recovers near-perfect discrimination where the threshold fails (ER, GEO); ensembles add negligible AUC but are extremely well calibrated.

	Threshold	LogReg	Rand. Forest	GBDT
AUC (by-graph generalization)				
BA	0.995	0.994	0.997	0.998
ER	0.826	0.998	1.000	1.000
GEO	0.890	0.992	0.998	0.999
Brier score (calibration)				
BA	—	0.024	0.013	0.012
ER	—	0.011	0.000	0.000
GEO	—	0.030	0.011	0.011

Three findings, all reinforcing the same point. *First*, on BA, the zero-ML threshold on $sz(c)$ (exact slate state, computed for free) already reaches AUC 0.995, matching every model; no learning is needed there at all. *Second*, where the threshold fails (ER, AUC 0.826), the gap is closed not by a nonlinear model but by a *linear* one: logistic regression jumps to 0.998, essentially matching the tree ensembles. The signal the threshold misses is not nonlinear—it is simply spread across the slate features in a way a linear combination

recovers. *Third*, gradient-boosted trees are best on every family but beat logistic regression by at most a few thousandths of AUC, while being extremely well calibrated (Brier ≤ 0.013): their predicted probabilities can be thresholded directly by the scheduler with confidence. The linear model also generalizes across families better than the ensembles (train BA, test GEO: 0.989 for logistic regression versus 0.965 GBDT, 0.941 random forest). The overall conclusion is deflationary and, we think, the honest one: on the favorable topology the scheduling signal is already in the slate, and where it is not, it is nearly *linearly* recoverable from slate state—ensembles and, a fortiori, heavier models such as graph neural networks are not required for this task.

7.4. Scheduling Value Under Bounded Staleness

A stream of 4000 updates under a fixed per-round budget with 15% headroom (a contended system), comparing FIFO, predictor-informed PRIORITY, and an ORACLE ceiling. PRIORITY improves mean time-to-resolution by 2.6× (BA), 3.8× (GEO), 2.1× (ER) over FIFO, concentrated on cheap updates, and matches ORACLE in every family—the result is not predictor-limited. Correctness is unaffected throughout (Proposition 5). Table 5 reports time-to-resolution (TTR, rounds from arrival to completion), and Figure 4 shows the mean-TTR improvement of PRIORITY over FIFO across the three families.

Table 5. Scheduling under contention. PRIORITY (predictor-informed) versus FIFO and an ORACLE ceiling; TTR in rounds. PRIORITY matches ORACLE throughout, so the result is not predictor-limited.

Fam.	Policy	Mean TTR	p90 TTR	Mean TTR (Cheap)
BA	FIFO	95.8	238	95.2
BA	PRIORITY	36.2	95	32.9
BA	ORACLE	36.2	95	32.6
GEO	FIFO	108.5	234	107.5
GEO	PRIORITY	28.2	69	21.5
ER	FIFO	43.3	118	42.7
ER	PRIORITY	20.8	65	18.4

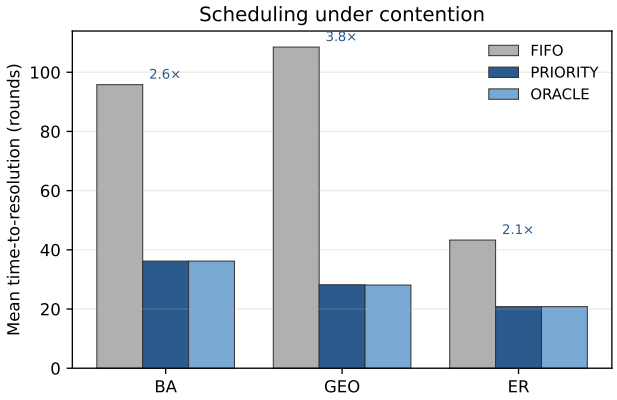

Figure 4. Mean time-to-resolution under contention. Predictor-informed PRIORITY scheduling improves over FIFO by the annotated factors and coincides with the ORACLE ceiling in every family, showing the benefit is not limited by predictor accuracy.

7.5. Threats to Validity

Case-B-triggered swaps are modeled as cost-1 in the scheduling stream (rare, bounded by the same machinery); the scheduling experiment tests one contention level, one graph and seed per family (the benefit vanishes without queueing pressure); the static comparison charges NTE its worst case against measured GHS and uses the synchronous GHS variant. These are conservative simplifications that do not affect the qualitative conclusions. Finally, our evaluation is simulation-based, measuring protocol-level message and round costs rather than wall-clock performance on deployed hardware. A testbed deployment on real IoT or overlay infrastructure is the natural next validation step and is discussed in Section 9.

8. Implementation Considerations

The protocol is specified in the synchronous CONGEST model; this section maps that specification onto deployable systems and quantifies the per-node resources it needs.

8.1. From Modules to Executables

Each of the six modules of Section 4.4 is an event handler triggered by a message arrival or a local timer, so the natural implementation is a single-threaded, message-driven event loop per node—the execution model native to embedded operating systems for constrained devices and to actor frameworks on server-class overlays. The slate is a per-node record; no shared memory, global clock, or topology map is required. The synchronous-round abstraction is realized either by the coarse time-slotting common in duty-cycled sensor MACs or, on asynchronous networks, by running the protocol under a synchronizer [50]; the α-synchronizer preserves round complexity at a constant-factor message overhead, which the worst-case charging of Section 7 already dominates. The model's reliable-link assumption maps to any transport with per-link reliable in-order delivery: TCP between overlay peers, or link-layer acknowledgment and retransmission (e.g., IEEE 802.15.4 [51] ARQ) on wireless fabrics.

8.2. Per-Node Resource Footprint

State. A node v stores its slate—identifiers and costs for its $\delta(v)$ incident edges, parent/child flags, the request sets, and the aggregation fields—totaling $O(\delta(v) \log n)$ bits, plus the maximum-weight label of Section 4.2 at $O(\log^2 n)$ bits under polynomial weights. Concretely, with 16-bit identifiers and 32-bit weights, a degree-8 sensor node stores under 100 bytes of slate and a label of order 64 bytes: comfortably within Class-1 constrained devices ($\sim$10 KB RAM) [52].

Messages. All three message types carry a constant number of fields (type tag, one edge identifier, one weight, one level), at most $\sim$12 bytes of payload with the encodings above; every message fits in a single IEEE 802.15.4 frame (127-byte MTU) or one UDP datagram, so the $O(\log n)$-bit bound of Claim 1 is not merely asymptotic. Network-wide, the static algorithm's worst case of $5m$ messages amounts to $\sim$120 KB of payload on a 1000-node Barabási–Albert fabric ($m \approx 2000$).

Processing. Every handler performs $O(\delta(v))$ integer comparisons per event; no floating point, cryptography, or dynamic allocation is required, and the degenerate-case handler is a constant-size guard.

8.3. Validation Path

The measured quantities of Section 7—messages and rounds—are the model-level proxies for transmission energy and latency respectively; converting them to joules and seconds for a concrete radio is a per-platform calibration. The natural validation sequence, planned as follow-up work (Section 9), is protocol emulation under a realistic radio and

duty-cycle model (e.g., Contiki-NG/COOJA), followed by deployment on a public wireless testbed (e.g., FIT IoT-LAB) for wireless fabrics and containerized overlay deployment for the P2P and SDN settings.

9. Conclusions and Future Work

For the infrastructures that motivated this work (sensor fabrics, content and learning overlays, software-defined networks) the operative question is not whether a minimum spanning tree can be *built*, but whether it can be *kept*: maintained correctly and cheaply as the network changes underneath it. This paper answers the algorithmic core of that question with a complete treatment: a fully specified and proven construction protocol, an exact maintenance algorithm whose cost is characterized at scale, a precise boundary on what any distributed approach can achieve, and a scheduling layer that turns prediction into operational value. Along the way we established that the most ambitious use of prediction—shortcutting the maintenance computation itself—is impossible in a precise sense, a boundary that itself became a contribution and that shaped the design toward scheduling.

On construction, infrastructures gain a protocol they can implement from specification: the NTE algorithm is now fully specified and proved (Section 4)—the complete module-level design, the Cycle Theorem and Theorem NTE, and proofs of the complexity claims ($O(m)$ messages of $O(\log n)$ bits in $O(d)$ rounds) appear in the peer-reviewed literature for the first time. Experimentally, the algorithm's *proven worst case* beats a measured synchronous GHS on rounds in every family tested (GHS taking 1.3–1.75$\times$ more) and on messages in the sparse regime ($\sim$2.2$\times$ fewer on Barabási–Albert graphs), with a density-dependent crossover on dense graphs—and the sparse regime is where the motivating applications live.

On maintenance—the operational core—the guarantees are proven and empirically characterized (Sections 5, 7): three of four update cases resolve in $O(1)$, and the fourth is resolved by ExactSwap within a bound shown essentially tight in practice (median ratio 1.00 to the true cost). That cost is favorably distributed exactly where it matters: on scale-free graphs the median affected set has size 2 *at every scale tested* from $n = 500$ to 20,000, a scale-invariance that considerably strengthens the case for the approach. The separation result (Proposition 4) delimits what any distributed algorithm can achieve here—exact replacement verification provably requires full-region search, a boundary with no centralized analogue—and directly shaped the final contribution: a scheduling-aware use of prediction that improves time-to-resolution 2.1–3.8$\times$ under contention while matching an oracle, and that needs less machine learning than one might expect, since the algorithm's own state largely suffices and the residual signal is nearly linearly recoverable.

Several directions follow. *The eager-caching question* (Section 5.5) is the most consequential open problem: caching a replacement per tree edge works cleanly against one invalidation trigger (proven), but a second—a Case-B weight decrease silently undercutting an existing cache entry—remains unbounded; resolving it would reopen a genuine cost-reduction guarantee in the style of van den Brand et al. [19], and this paper has now located precisely where the difficulty lies. *Edge insertions and deletions* are treated in Section 5.4: the insertion decision is already an $O(1)$ consequence of existing state (Proposition 3), deletion reduces to Case-D machinery, and a full treatment under realistic churn—mechanism, correctness, and evaluation together—is left to future work. *A measured NTE simulator* would replace the worst-case charging of Section 7 with measured message counts, sharpening the dense-graph crossover in NTE's favor. *Robustness sweeps* for the scheduling result, across contention levels, graphs, and streams, are inexpensive with the pipeline in place; the benefit necessarily vanishes without queueing pressure, and mapping where it turns on is

worth reporting. Finally, the three infrastructure settings that motivated this work each map directly onto the machinery developed here, with a distinct cost currency in each case. The readings below are protocol-level: they translate the measured message and round costs into each setting's operational currency, and hold subject to the per-platform calibration described in Section 8. *Energy-constrained sensor fabrics* [1,2]: node death from battery depletion is precisely a Case-D-like event, and the scale-invariant median affected set of 2 means the energy cost of repairing the aggregation tree does not grow with deployment size on scale-free topologies—the currency is energy, and the O(1) resolution of Cases A–C means routine link-quality drift costs essentially nothing. *Decentralized federated-learning overlays* [4]: participant churn between training rounds is a stream of well-separated structural events, exactly the update model of Section 3—and the aggregation topology whose placement recent work optimizes [44] is precisely the structure our algorithm keeps minimal between placements— and the scheduling layer of Section 6 lets overlay repair be deferred to inter-round windows—the currency is training-round latency, and the oracle-matching PRIORITY policy shows that the deferral decision needs no heavyweight learning infrastructure. *Software-defined multicast fabrics* [6]: link failure triggers tree repair whose cost the controller must budget, and the tight per-update bound (median ratio 1.00) gives the controller an accurate, locally-computable cost estimate before committing to a repair—the currency is reconfiguration latency. A testbed evaluation in one of these settings, replacing our protocol-level simulation with deployed-infrastructure measurements, is the natural next step and would complete the arc from formal specification to operational tool.

Funding: This research was supported by the Deanship of Scientific Research, Vice Presidency for Graduate Studies and Scientific Research, King Faisal University, Al Ahsa, Saudi Arabia [Grant number KFU264054].

Data Availability Statement: The datasets and seeded simulation code reproducing all experiments in Section 7 are available from the author on request.

Acknowledgments: During the preparation of this work, the author used Claude Opus 4.1 (Anthropic) to assist with language editing, improving the clarity of the manuscript, and preparing the LaTeX source. AI-assisted output was carefully reviewed, revised, and verified by the author. The author directed and reviewed AI-assisted output, verified technical content, and takes full responsibility for the content of the published article.

Conflicts of Interest: The author declares no conflict of interest.

Appendix A. Detailed Pseudocode

The five modules below complete the specification begun with Algorithm 1 (the update aggregator, in the main text). All listings have been validated against the simulated trace of Section 4.5.

Algorithm A1 Request Generator (at the session initiator)

1: **on** initialization:
2: $level \leftarrow 1$; designate $update_initiator \in \mathrm{nbr}(self)$
3: **for all** $y \in \mathrm{nbr}(self)$ **do**
4: send REQUEST$(\mathrm{id}(self), \mathrm{id}(y), level, \mathrm{id}(self), update_initiator, sid)$ to y
5: **end for**

Algorithm A2 Request Aggregator / Propagator (at node y, on REQUEST from x)

if $level_y = 0$ **then** ▷ unleveled sentinel, Equation (1)
  $level_y \leftarrow level_x + 1$; $rr(y) \leftarrow rr(y) \cup \{x\}$
  if x qualifies as parent(y) by Equation (2) **then**
    parent(y) $\leftarrow x$; send REPLY(y, x, ACK)
  else
    send REPLY(y, x, NACK)
  end if
  for all $z \in$ nbr(y) $\setminus$ {parent(y)} **do**
    $rs(y) \leftarrow rs(y) \cup \{z\}$; send REQUEST($y, z, level_y, S, D, sid$) to z
  end for
else
  resolve parent conflict via Equation (2) only; reply ACK/NACK accordingly
end if

Algorithm A3 Reply Aggregator (at node x)

on REPLY(y, x, ACK): child(x) $\leftarrow$ child(x) $\cup \{y\}$
on REPLY(y, x, NACK): no action
on all expected replies received: **if** $x = D$ **then** invoke Update Generator

Algorithm A4 Update Generator (at the update initiator D; D has no children)

for all $p \in$ parents of D in the orientation **do**
  send UPDATE(D, p, *cand*=(D, p), *maxlink*=cost(D, p), *nscand*=$\emptyset$, *nsmax*=0) to p
end for

Algorithm A5 Degenerate-Case Handler

if node u_0 received requests from *every* neighbor (null update initiator):
  mark every incident edge except the minimum-cost one as NTE, locally, with no update messages
  send null *maxlink*/*nsmax* values to all parents
if a node has exactly one child: forward the child's update values unchanged (single-child case of Algorithm 1)

Note (Remark 1): the field *candidate_weight* is written by none of these modules; that field is populated only by the post-construction labeling pass described after Remark 1.

References

1. Akyildiz, I.F.; Su, W.; Sankarasubramaniam, Y.; Cayirci, E. Wireless Sensor Networks: A Survey. *Comput. Netw.* **2002**, *38*, 393–422. [CrossRef]
2. Njoya, A.N.; Thron, C.; Awa, M.N.; Ari, A.A.A.; Gueroui, A.M. Lifetime Optimization of Dense Wireless Sensor Networks Using Continuous Ring-Sector Model. *Future Gener. Comput. Syst.* **2022**, *129*, 212–224. [CrossRef]
3. Chu, Y.h.; Rao, S.G.; Seshan, S.; Zhang, H. A Case for End System Multicast. *IEEE J. Sel. Areas Commun.* **2002**, *20*, 1456–1471. [CrossRef]
4. Kairouz, P.; McMahan, H.B. Advances and Open Problems in Federated Learning. *Found. Trends Mach. Learn.* **2021**, *14*, 1–210. [CrossRef]
5. Prigent, C.; Costan, A.; Antoniu, G.; Cudennec, L. Enabling Federated Learning Across the Computing Continuum: Systems, Challenges and Future Directions. *Future Gener. Comput. Syst.* **2024**, *160*, 767–783. [CrossRef]
6. Kreutz, D.; Ramos, F.M.V.; Veríssimo, P.E.; Rothenberg, C.E.; Azodolmolky, S.; Uhlig, S. Software-Defined Networking: A Comprehensive Survey. *Proc. IEEE* **2015**, *103*, 14–76. [CrossRef]
7. Gallager, R.G.; Humblet, P.A.; Spira, P.M. A Distributed Algorithm for Minimum-Weight Spanning Trees. *ACM Trans. Program. Lang. Syst.* **1983**, *5*, 66–77. [CrossRef]

8. Awerbuch, B. Optimal Distributed Algorithms for Minimum Weight Spanning Tree, Counting, Leader Election, and Related Problems. In *Proceedings of the 19th Annual ACM Symposium on Theory of Computing (STOC)*; Association for Computing Machinery: New York, NY, USA, 1987; pp. 230–240.
9. Garay, J.A.; Kutten, S.; Peleg, D. A Sublinear Time Distributed Algorithm for Minimum-Weight Spanning Trees. *SIAM J. Comput.* **1998**, *27*, 302–316. [CrossRef]
10. Faloutsos, M.; Molle, M. A Linear-Time Optimal-Message Distributed Algorithm for Minimum Spanning Trees. *Distrib. Comput.* **2004**, *17*, 151–170. [CrossRef]
11. Lotker, Z.; Patt-Shamir, B.; Pavlov, E.; Peleg, D. Minimum-Weight Spanning Tree Construction in $O(\log \log n)$ Communication Rounds. *SIAM J. Comput.* **2005**, *35*, 120–131. [CrossRef]
12. Pandurangan, G.; Robinson, P.; Scquizzato, M. A Time- and Message-Optimal Distributed Algorithm for Minimum Spanning Trees. *ACM Trans. Algorithms* **2019**, *16*, 1–27. [CrossRef]
13. Ragab, K.; Haque, A.U. A Minimum Spanning Tree Algorithm for Efficient P2P Video Streaming System. In Proceedings of the International Conference on Advanced Communication Technology (ICACT), Gangwon, Republic of Korea, 7–10 February 2010.
14. Khan, J.I.; Haque, A.U. Maximizing Lifetime of Power-Limited Network with Active Minimum Spanning Tree Aggregation. In *Proceedings of the 17th International Conference on Parallel and Distributed Computing and Systems (PDCS)*; ACTA Press: Cambridge, MA, USA, 2004.
15. Díaz, S.; Méndez, D.; Schölzel, M. Dynamic Minimum Spanning Tree Construction and Maintenance for Wireless Sensor Networks. *Rev. Fac. Ing. Univ. Antioq.* **2019**, *93*, 57–69. [CrossRef]
16. Xu, L.; Wen, D.; Qin, L.; Li, R.; Zhang, Y.; Lu, Y.; Lin, X. Minimum Spanning Tree Maintenance in Dynamic Graphs. *Proc. Acm Manag. Data (PACMMOD)* **2025**, *3*, 1–24. [CrossRef]
17. Lykouris, T.; Vassilvitskii, S. Competitive Caching with Machine Learned Advice. *J. ACM* **2021**, *68*, 1–25. [CrossRef]
18. Mitzenmacher, M.; Vassilvitskii, S. Algorithms with Predictions. *Commun. ACM* **2022**, *65*, 33–35. [CrossRef]
19. van den Brand, J.; Forster, S.; Nazari, Y.; Polak, A. On Dynamic Graph Algorithms with Predictions. In *Proceedings of the ACM-SIAM Symposium on Discrete Algorithms (SODA)*; Society for Industrial and Applied Mathematics: Philadelphia, PA, USA, 2024.
20. Haque, A.U. Bounded Wait Communication for Systems with Distributed and Autonomous Components. Ph.D. Thesis, Kent State University, Kent, OH, USA, 2006.
21. Chin, F.; Ting, H.F. An Almost Linear Time and $O(V \log V + E)$ Messages Distributed Algorithm for Minimum-Weight Spanning Trees. In Proceedings of the 26th IEEE Symposium on Foundations of Computer Science (FOCS), Portland, OR, USA, 21–23 October 1985; pp. 257–266.
22. Peleg, D. *Distributed Computing: A Locality-Sensitive Approach*; SIAM: Philadelphia, PA, USA, 2000.
23. Kutten, S.; Peleg, D. Fast Distributed Construction of Small k-Dominating Sets and Applications. *J. Algorithms* **1998**, *28*, 40–66. [CrossRef]
24. Peleg, D.; Rubinovich, V. A Near-Tight Lower Bound on the Time Complexity of Distributed Minimum-Weight Spanning Tree Construction. *SIAM J. Comput.* **2000**, *30*, 1427–1442. [CrossRef]
25. Elkin, M. A Simple Deterministic Distributed MST Algorithm, with Near-Optimal Time and Message Complexities. In Proceedings of the ACM Symposium on Principles of Distributed Computing (PODC), Washington, DC, USA, 25–27 July 2017; pp. 157–163.
26. Khan, M.; Pandurangan, G. A Fast Distributed Approximation Algorithm for Minimum Spanning Trees. *Distrib. Comput.* **2008**, *20*, 391–402. [CrossRef]
27. Huang, S.T. A Fully Pipelined Minimum-Spanning-Tree Constructor. *J. Parallel Distrib. Comput.* **1990**, *9*, 55–62. [CrossRef]
28. Sleator, D.D.; Tarjan, R.E. A Data Structure for Dynamic Trees. *J. Comput. Syst. Sci.* **1983**, *26*, 362–391. [CrossRef]
29. Karger, D.R.; Klein, P.N.; Tarjan, R.E. A Randomized Linear-Time Algorithm to Find Minimum Spanning Trees. *J. ACM* **1995**, *42*, 321–328. [CrossRef]
30. Frederickson, G.N. Data Structures for On-Line Updating of Minimum Spanning Trees, with Applications. *SIAM J. Comput.* **1985**, *14*, 781–798. [CrossRef]
31. Eppstein, D.; Galil, Z.; Italiano, G.F.; Nissenzweig, A. Sparsification—A Technique for Speeding Up Dynamic Graph Algorithms. *J. ACM* **1997**, *44*, 669–696. [CrossRef]
32. Holm, J.; de Lichtenberg, K.; Thorup, M. Poly-Logarithmic Deterministic Fully-Dynamic Algorithms for Connectivity, Minimum Spanning Tree, 2-Edge, and Biconnectivity. *J. ACM* **2001**, *48*, 723–760. [CrossRef]
33. Eppstein, D.; Italiano, G.F.; Tamassia, R.; Tarjan, R.E.; Westbrook, J.; Yung, M. Maintenance of a Minimum Spanning Forest in a Dynamic Plane Graph. *J. Algorithms* **1992**, *13*, 33–54. [CrossRef]
34. Luo, M.; Qin, H.; Wu, X.; Xiong, C.; Xia, D.; Ke, Y. Efficient Maintenance of Minimum Spanning Trees in Dynamic Weighted Undirected Graphs. *Mathematics* **2024**, *12*, 1021. [CrossRef]
35. Awerbuch, B.; Cidon, I.; Kutten, S. Optimal Maintenance of a Spanning Tree. *J. ACM* **2008**, *55*, 1–45. [CrossRef]

36. Gupta, S.K.S.; Srimani, P.K. Self-Stabilizing Multicast Protocols for Ad Hoc Networks. *J. Parallel Distrib. Comput.* **2003**, *63*, 87–96. [CrossRef]
37. Peleg, D. Distributed Matroid Basis Completion via Elimination Upcast and Distributed Correction of Minimum-Weight Spanning Trees. In *Proceedings of the 25th International Colloquium on Automata, Languages and Programming (ICALP)*; Springer: Berlin/Heidelberg, Germany, 1998; pp. 164–175.
38. Foerster, K.T.; Korhonen, J.H.; Paz, A.; Rybicki, J.; Schmid, S. Input-Dynamic Distributed Algorithms for Communication Networks. *Proc. ACM Meas. Anal. Comput. Syst.* **2021**, *5*, 1–33. [CrossRef]
39. Purohit, M.; Svitkina, Z.; Kumar, R. Improving Online Algorithms via ML Predictions. In Proceedings of the Advances in Neural Information Processing Systems (NeurIPS), Montréal, QC, Canada, 3–8 December 2018; pp. 9684–9693.
40. Abdel-Wahab, H.; Stoica, I.; Sultan, F.; Wilson, K. A Simple Algorithm for Computing Minimum Spanning Trees in the Internet. *Inf. Sci.* **1997**, *101*, 47–69. [CrossRef]
41. Hossain, M.A.; Khan, J.I. ZePoP: A Distributed Leader Election Protocol using the Delay-based Closeness Centrality for Peer-to-Peer Applications. In Proceedings of the 12th IEEE International Conference on Cloud Networking (CloudNet), Hoboken, NJ, USA, 1–3 November 2023. [CrossRef]
42. Mokhtarian, K.; Jacobsen, H.A. Minimum-Delay Overlay Multicast. In Proceedings of the IEEE INFOCOM, Turin, Italy, 14–19 April 2013; pp. 1771–1779. [CrossRef]
43. Mohammed, N.A.; Othman, M. A Load-Balanced Algorithm for Internet Gateway Placement in Backbone Wireless Mesh Networks. *Future Gener. Comput. Syst.* **2024**, *150*, 144–159. [CrossRef]
44. Xu, B.; Zhao, H.; Cao, H.; Garg, S.; Kaddoum, G.; Hassan, M.M. Edge Aggregation Placement for Semi-Decentralized Federated Learning in Industrial Internet of Things. *Future Gener. Comput. Syst.* **2024**, *150*, 160–170. [CrossRef]
45. Zhang, Y.; Duffield, N.; Paxson, V.; Shenker, S. On the Constancy of Internet Path Properties. In Proceedings of the ACM SIGCOMM Internet Measurement Workshop (IMW), San Francisco, CA, USA, 1–2 November 2001; pp. 197–211.
46. Paxson, V. End-to-End Routing Behavior in the Internet. *IEEE/ACM Trans. Netw.* **1997**, *5*, 601–615. [CrossRef]
47. Korman, A.; Kutten, S. Distributed Verification of Minimum Spanning Trees. *Distrib. Comput.* **2007**, *20*, 253–266. [CrossRef]
48. Cormen, T.H.; Leiserson, C.E.; Rivest, R.L.; Stein, C. *Introduction to Algorithms*, 4th ed.; MIT Press: Cambridge, MA, USA, 2022.
49. Prim, R.C. Shortest Connection Networks and Some Generalizations. *Bell Syst. Tech. J.* **1957**, *36*, 1389–1401. [CrossRef]
50. Awerbuch, B. Complexity of Network Synchronization. *J. ACM* **1985**, *32*, 804–823. [CrossRef]
51. *IEEE Std 802.15.4-2024*; IEEE Standard for Low-Rate Wireless Networks. IEEE: New York, NY, USA, 2024.
52. Bormann, C.; Ersue, M.; Keränen, A. *Terminology for Constrained-Node Networks*; RFC 7228; Internet Engineering Task Force: Wilmington, DE, USA, 2014.

Disclaimer/Publisher's Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.