Improved and Deterministic Online Service with Deadlines or Delay

Noam Touitou
Amazon
Tel Aviv, Israel
noamtwx@gmail.com

ABSTRACT

We consider the problem of online service with delay on a general metric space, first presented by Azar, Ganesh, Ge and Panigrahi (STOC 2017). The best known randomized algorithm for this problem, by Azar and Touitou (FOCS 2019), is $O(\log^2 n)$-competitive, where n is the number of points in the metric space. This is also the best known result for the special case of online service with deadlines, which is of independent interest.

In this paper, we present $O(\log n)$-competitive *deterministic* algorithms for online service with deadlines or delay, improving upon the results from FOCS 2019. Furthermore, our algorithms are the first deterministic algorithms for online service with deadlines or delay which apply to general metric spaces and have sub-polynomial competitiveness.

CCS CONCEPTS

• **Theory of computation** → **Online algorithms**; **K-server algorithms**.

KEYWORDS

online, deadlines, delay, service, k-server

ACM Reference Format:
Noam Touitou. 2023. Improved and Deterministic Online Service with Deadlines or Delay. In *Proceedings of the 55th Annual ACM Symposium on Theory of Computing (STOC '23), June 20–23, 2023, Orlando, FL, USA.* ACM, New York, NY, USA, 14 pages. https://doi.org/10.1145/3564246.3585107

1 INTRODUCTION

In online service with deadlines/delay, a server exists on a metric space of n points. Requests arrive over time on points in the metric space, demanding service by the algorithm. The algorithm can serve requests by moving the server to their location, incurring a cost which is the distance traveled by the server on the metric space. In *online service with deadlines*, each request has an associated deadline by which it must be served. In *online service with delay*, a more general problem, the deadline is replaced with delay costs which accrue while the request is pending. Specifically, each request has an associated, non-decreasing delay function, such that the total delay cost incurred by a pending request until time t is the value of its delay function at t.

Online service with delay was first introduced by Azar et al. [1], who gave an $O(\log^4 n)$ randomized algorithm for the problem, based on randomized embedding of the metric space into a tree (specifically, a weighted hierarchically well-separated tree), then solving the problem on the resulting tree. In [2], this was improved to $O(\log^2 n)$-competitiveness, through an improved algorithm for online service on a tree. The result of [2] remains the best known randomized result for this problem.

Without randomization, much less is known about this problem. There is no known deterministic algorithm (of competitiveness less than polynomial) which applies to general metric spaces. For *specific* metric spaces, some results are known. When the metric space is uniform (or weighted uniform), the work of Azar et al. [1] implies a constant-competitive deterministic algorithm. When the metric space is a line, Bienkowski et al. [7] presented an $O(\log \Delta)$-competitive deterministic algorithm; here, Δ is the aspect ratio of the metric space, or the ratio between the largest and smallest pairwise distances (for a line, note that $\Delta \geq n$).

1.1 Our Results

We consider online service with deadlines/delay on a metric space of n points, and present the following results.

(1) An $O(\log n)$-competitive, deterministic algorithm for online service with deadlines that runs in polynomial time. (Section 3.)
(2) An $O(\log n)$-competitive, deterministic algorithm for online service with delay that runs in polynomial time. (Appendix A.)

Both results improve upon the best known *randomized* algorithm for service with deadlines/delay, which is the randomized $O(\log^2 n)$-competitive algorithm of [2]. Moreover, these are the first deterministic algorithms of sub-polynomial competitiveness for online service with deadlines/delay on general metric spaces. Note that while the result for deadlines is implied by the result for delay, we chose to present it independently. This is both for ease of presentation and since the deadline case is of independent interest.

In fact, we show that our algorithms achieve a stronger result: they are $O(\log \min\{n, m\})$-competitive, where m is the number of requests in the input. Note that previous algorithms had no guarantee in terms of the number of requests. Specifically, previous algorithms were based on randomized tree embedding, and thus lose $\Theta(\log n)$ in competitiveness even when m is constant. We discuss this result in the full version of the paper.

STOC '23, June 20–23, 2023, Orlando, FL, USA

ACM ISBN 978-1-4503-9913-5/23/06…$15.00
https://doi.org/10.1145/3564246.3585107

1.2 Our Techniques

Online service with deadlines.. The algorithm for online service with deadlines employs the main concept used in [3] for network design problems with deadlines, which is to assign levels to requests that increase over time, such that high-level requests are only served in high-cost services. Services also have levels, determined by the level of request whose deadline has been reached. The budget of a service is exponential in its level, and a service of level ℓ only serves requests of level at most ℓ.

However, in network design the cost of serving a request is fixed at all times, while in online service this cost depends on the current location of the server. Thus, while levels are maintained for each request, the participation of a request in a service depends both on its level and on its distance from the server at the time of service; the resulting parameter is called the *adjusted level* of the request. As a service of level ℓ restricts itself to requests with adjusted level at most ℓ, each service is confined to some "service ball" centered at the server's location.

In addition, online service calls for a more aggressive raising of levels. In particular, the following properties are crucial to the analysis:

(1) Upon the deadline of a request, a service is started with level much larger than that of the triggering request.
(2) Upon the end of a service, requests in its service ball are upgraded to a *higher* level than that of the service.

Compare this to the network design framework of [3], in which the constant difference in levels between service and triggering request can be any positive number without breaking the analysis (and is chosen to be 1). In addition, for network design the level of an eligible request only increased to the level of the service.

Finally, the server itself must occasionally move to more opportune locations. In our algorithm, this depends on the triggering request: if its adjusted level is dictated by its distance from the server, we say that the service is *primary*. In this case, the server is moved to the request at the end of the service; otherwise, the server returns to its initial position.

Analysis. The optimal solution for online service is harder to characterize than in network design. Optimal services in network design can be charged independently (where the requests served are an "intersecting set"), while the tour of the optimal server in online service is not easily partitioned. This calls for a novel type of analysis that we introduce.

In our analysis, we construct space-time *cylinders*, where each cylinder is associated with a shape in the metric space (e.g., a ball) and a time interval. We then show two properties: first, that the optimal solution incurs enough cost inside each cylinder; second, that the cylinders are disjoint (either temporally or spatially). This yields a lower bound on the cost of the optimal solution; as each cylinder is associated with a service in the algorithm, this connects the cost of the algorithm with that of the optimal solution. The aggressive upgrading of requests and services is dictated by this analysis: upgrading requests forces OPT to incur high cost inside each cylinder, while upgrading services implies disjointness of the cylinders. We first use cylinders in a simple way, such that the cylinders' associated shapes are balls in the metric space. Then, to show our final result, we perforate those balls by removing balls of much-smaller radius; we then show that the charge to the optimum is maintained, while achieving a greater degree of cylinder disjointness (and thus a better competitive ratio).

Online service with delay. The algorithm for online service with delay is similar to the algorithm for deadlines. In deadlines, a service is started when the deadline of a request expires; in delay, a service is started when the total delay for a set of pending requests becomes large. (In fact, we consider the *residual* delay for this condition, which is the amount of delay that exceeds investment from past services.) Specifically, when requests of adjusted level at most ℓ gather a total delay of 2^{ℓ}, we say that level ℓ has become critical and trigger a service. This service uses a *prize-collecting* algorithm for Steiner tree to choose whether to serve requests or invest in them, offsetting their future delay. This use of a prize-collecting approximation algorithm is similar to that in [3].

The salient difference between the deadline and delay case lies in moving the server upon a primary service (where "primary" is defined somewhat similarly to deadlines). Considering deadlines as a specific case of delay, an expired deadline is equivalent to infinite delay incurred in a very concentrated neighborhood (i.e., a single point). Analogously, for the case of delay, in a primary service we attempt to identify a small-radius ball within which a constant fraction of the residual delay exists. If such a ball is found, the server would move to its center at the end of the service. Otherwise, the delay is well-spread, and the server would remain stationary. The intuition here is that when the delay is well-spread, the optimal solution must also make significant movements to avoid incurring large delay.

1.3 Related Work

Multiple servers. Online service with delay has also been considered when the algorithm has $k > 1$ servers. In the first paper of Azar et al. [1], an $O(k \cdot \text{poly}\log(n))$-competitive randomized algorithm was given for this problem. As the algorithm only used randomization in the initial embedding stage, its dependence on k is linear (as online service with delay is a generalization of the k-server problem, which has an $\Omega(k)$-competitiveness lower bound for deterministic algorithms). For uniform metric spaces, a better use of randomization was done in [16], achieving an $O(\log n \log k)$-competitive algorithm. For online service with deadlines on a general metric space, an $O(\text{poly}\log(\Delta n))$-competitive randomized algorithm was presented by Gupta et al. [15].

Network design with deadlines/delay. A set of problems related to online service is network design with deadlines/delay. In such problems, connectivity requests with deadlines or delay arrive over time, and must be served by transmitting a subgraph that provides the desired connectivity. A notable example is Steiner tree, in which requests demand connecting a terminal to some root node. It can be seen that Steiner tree with deadlines/delay is a special case of online service with deadlines/delay: the reduction involves forcing the server to remain at the root node through a stream of requests with immediate deadline (or very high delay cost). The special case of Steiner tree in which the metric space is itself a tree is called multilevel aggregation, and has received much attention [2, 5, 10]; this is true also of its special cases, TCP acknowledgement (e.g., [11, 14, 18]) and joint replenishment ([6, 8, 12]). Multilevel aggregation itself has also yielded algorithms for online service with delay (see [2]). A general framework for network design problems was introduced in [3]; some techniques introduced in [3] are used in this paper for online service. These techniques were also used by Chen et al. [13] for a generalization of joint replenishment.

Classic online k-server. In k-server, the classic variant of online service with deadlines/delay, requests arrive over a sequence rather than over time to be served by one of k servers. (Here, unlike in service with deadlines/delay, the case of a single server is trivial.) Deterministically, the best competitiveness bound for this problem is $\Theta(k)$ [20, 21], where determining the exact constant is an open problem. With randomization, poly-logarithmic competitive ratios have been achieved relatively recently [4, 9].

2 PRELIMINARIES

In online service with deadlines/delay, we are given a metric space of n points. We represent this metric space as a weighted, simple graph G of n nodes, such that the distance $\delta(u, v)$ between two points in the metric space is the weight of the shortest path between the nodes in the graph. Each request q in the input request set Q arrives at time r_q; slightly abusing notation, we also use q to denote the point in G on which the request exists. A server exists in the metric space, such that moving the server to a pending request q serves the request (the server movements are immediate, and do not require time).

In the deadline case, each request $q \in Q$ has deadline d_q, and must be served in the interval $(r_q, d_q]$; we assume WLOG that the deadlines of all requests are distinct (this can be enforced by arbitrary tie breaking by the algorithm). The goal is to minimize the total movement of the server during the course of the algorithm, while still serving all requests by their deadline.

In the delay case, each request $q \in Q$ has a nondecreasing delay function $d_q(t)$, defined for every time $t \geq r_q$, such that the total delay cost that pending request q accrues by time t is $d_q(t)$.

Without loss of generality, we assume that $d_q(r_q) = 0$, and that delay rises continuously. (Indeed, the former assumption translates to an additive constant to every solution, while the latter can again be enforced by the algorithm.) For ease of presentation, and in keeping with some previous work, we also assume that the delay of every request tends to infinity as time advances; that is, that every request must be served eventually. (We remark that our algorithm can also be seen to work without this assumption.)

For every number x, we define $(x)^+ := \max\{x, 0\}$. Given a point $v \in G$ and a radius r, we define $B(v, r)$ to be the set of nodes $u \in G$ such that $\delta(v, u) \leq r$.

3 ONLINE SERVICE WITH DEADLINES

In this section, we consider online service with deadlines, and prove the following theorem.

Theorem 3.1. *There exists an $O(\log n)$-competitive deterministic algorithm for online service with deadlines.*

3.1 The Algorithm

Steiner tree. Our algorithm contains a component which produces an approximate solution to the (offline) Steiner tree problem. In this problem, one is given a set of terminal nodes in a graph, and must output a minimum-cost subtree spanning those terminals. A classic result for approximating offline Steiner tree [19] shows that there exists a 2-approximation for this problem; we denote this approximation algorithm by ST, such that $\mathrm{ST}(U)$ denotes the output of the algorithm on the graph G given the set of terminals U. Slightly abusing notation, we also use $\mathrm{ST}(U)$ to denote the *cost* of the approximate solution. Similarly, we use $\mathrm{ST}^*(U)$ to denote some optimal solution for Steiner tree on terminals U (or its cost).

Algorithm's description. We now describe the behavior of the algorithm for online service with deadlines. The algorithm is divided into *services*, which are instantaneous events in which the algorithm decides to move its server to serve requests. For every pending request q, the algorithm maintains a level ℓ_q, which limits the set of services for which the request can be eligible (initially, $\ell_q = -\infty$). In addition, with a the current location of the server, we define the *adjusted level* of a request q to be

$$\bar{\ell}_q := \max\{\ell_q, \lceil \log \delta(a, q) \rceil\}$$

Upon deadline of request q, the algorithm starts a new service λ. The service λ also has a level ℓ_λ, which is larger by a constant from the adjusted level of the triggering request q. The level of λ determines which pending requests are considered for service by λ; specifically, a request q' is eligible for service only if $\bar{\ell}_{q'} \leq \ell_\lambda$. This means that λ restricts itself to requests that are both of level at most ℓ_λ, and are within the ball $B(a, 2^{\ell_\lambda})$, where a is the current location of the server.

Once the eligible requests have been identified, the algorithm attempts to solve them by order of increasing deadlines, subject to a budget of $\Theta(2^{\ell_\lambda})$. This makes use of a Steiner tree approximation component, to design an efficient path through the chosen requests. The algorithm then traverses this path, serving the chosen requests and finishing at its starting position a. For the remaining, unserved eligible requests, their level is raised to *above* the level of the service; specifically, to level $\ell_\lambda + 1$.

Finally, note that for the triggering request q, $\bar{\ell}_q$ is dictated by either ℓ_q or $\delta(q, a)$. If it is dictated by the latter, the service is called a *primary* service, and the service would move the server from a to q. Otherwise, the server would remain at a at the end of the service. The pseudocode description of the algorithm is given in Algorithm 1.

3.2 Analysis

Our goal now is to analyze Algorithm 1 and prove Theorem 3.1. Recall that we denote by Δ the aspect ratio of the metric space, i.e., the ratio between the largest and smallest pairwise distances. For ease of exposition, we first prove the following weaker theorem;

Theorem 3.2 (weaker version of Theorem 3.1). *There exists an $O(\log(n\Delta))$-competitive deterministic algorithm for online service with deadlines.*

After proving Theorem 3.2, we show how to strengthen some components in the analysis to obtain Theorem 3.1.

Basic Definitions and Properties. We denote by Λ the set of services performed by the algorithm.

Algorithm 1: Online Service with Deadlines

```
1  Event Function UponRequest($q$)
2      set $\ell_q \leftarrow -\infty$.
3  Event Function UponDeadline($q$)
4      start a new service, denoted by $\lambda$.
5      let $a$ be the current location of the server.
6      if $\bar{\ell}_q \neq \ell_q$ then say $\lambda$ is primary else $\lambda$ is not primary.
7      set service level $\ell_\lambda \leftarrow \bar{\ell}_q + 3$.
8      let $Q'$ be the set of currently pending requests.
9      let $E_\lambda \leftarrow \{q' \in Q' | \bar{\ell}_{q'} \leq \ell_\lambda\}$.
10     let $Q_\lambda \leftarrow \{q\}, S \leftarrow \emptyset$.
11     for $q' \in E_\lambda$ by order of increasing deadline do
12         add $q'$ to $Q_\lambda$.
13         let $S \leftarrow \mathrm{ST}(Q_\lambda)$.
14         if $c(S) \geq 4 \cdot 2^{\ell_\lambda}$ then
15             break from the loop.
16     perform DFS tour of $S$, serving $Q_\lambda$.
17     foreach $q' \in E_\lambda \setminus Q_\lambda$ do
18         set $\ell_{q'} \leftarrow \ell_\lambda + 1$.
19     if $\lambda$ is primary then move the server to $q$.
```

Definition 3.3 (basic service definitions). Let $\lambda \in \Lambda$ be a service. We define:

- The *triggering request* of λ, denoted $q^\star_\lambda$, to be the request whose deadline started λ.
- The location a_λ to be the initial location of the server when λ is triggered.
- The service time $t_\lambda := d_{q^\star_\lambda}$.
- The request set Q_λ to be the requests served by λ (i.e., the final value of the variable of that name in UponDeadline).
- The request set E_λ, as defined in Line 9; these requests are called *eligible for* λ.
- The *forwarding time* of λ, denoted τ_λ, to be the maximum deadline of a request in Q_λ.
- The *cost* of λ, denoted by $c(\lambda)$, to be the total cost of moving the server in λ. For a set of services Λ', we define $c(\Lambda') := \sum_{\lambda \in \Lambda'} c(\lambda)$.

We now define two subsets of services which are of particular focus: *primary* services and *certified* services. (Note that a service can belong to both subsets.) We later show that bounding the costs of these two subsets is enough to bound the total cost of the algorithm.

Definition 3.4 (primary services). A service $\lambda \in \Lambda$ is called *primary* if it is set to be primary in Line 6; that is, if $\bar{\ell}_{q^\star_\lambda} \neq \ell_{q^\star_\lambda}$ at t_λ. We denote by $\Lambda^{\mathrm{p}} \subseteq \Lambda$ the set of primary services in the algorithm.

Definition 3.5 (witness requests and certified services). At a certain point in time, a request q is called a *witness* for a service λ if its level ℓ_q was last modified by λ (at Line 18).

Note that the triggering request of a non-primary service λ' is always a witness for an earlier service λ; we say that λ' *certifies* λ, and call λ a *certified* service. We denote by $\Lambda^{\mathrm{c}} \subseteq \Lambda$ the set of certified services in the algorithm.

Proposition 3.6. *Every certified service $\lambda \in \Lambda^{\mathrm{c}}$ is certified by exactly one other service.*

Proof. Suppose, for contradiction that λ is certified by two services, λ_1, λ_2, and assume WLOG that $t_{\lambda_1} < t_{\lambda_2}$. It must be that the triggering requests $q^\star_{\lambda_1}, q^\star_{\lambda_2}$ were witnesses for λ at $t_{\lambda_1}, t_{\lambda_2}$, respectively. Thus, these requests were also of level $\ell_\lambda + 1$, which implies $\ell_{\lambda_1} = \ell_{\lambda_2} = \ell_\lambda + 4$.

We claim that after λ_1, there remain no witness requests for λ, in contradiction to $q^\star_{\lambda_2}$ being such a witness. Indeed, consider the state immediately before λ_1: all witness requests for λ at that time have level $\ell_\lambda + 1$, and exist within the ball $B\left(a_\lambda, 2^{\ell_\lambda}\right)$ (as they were eligible for λ). Both $q^\star_{\lambda_1}, q^\star_{\lambda_2}$ are witness requests for λ at that time, and thus: Thus,

$$\delta\left(a_{\lambda_1}, q^\star_{\lambda_2}\right) \leq \delta\left(a_{\lambda_1}, q^\star_{\lambda_1}\right) + \delta\left(q^\star_{\lambda_1}, a_\lambda\right) + \delta\left(a_\lambda, q^\star_{\lambda_2}\right)$$
$$\leq 2^{\ell_{\lambda_1} - 3} + 2^{\ell_\lambda} + 2^{\ell_\lambda} \leq 2^{\ell_{\lambda_1}}.$$

Therefore, $q^\star_{\lambda_2} \in E_{\lambda_1}$; note that all requests in E_{λ_1} are either served in λ_1 or become witnesses for λ_1. □

Proposition 3.7. *The cost of a service λ is at most $O(1) \cdot 2^{\ell_\lambda}$.*

Proof. First, observe the cost of traversing the Steiner tree solution for the set of requests Q_λ as formed in the algorithm. Note that a possible Steiner tree solution for connecting $Q_\lambda \cup \{a_\lambda\}$ would be to use the Steiner tree solution calculated in the penultimate iteration of the loop to connect all requests except q', where q' is the final request added to Q_λ, then connect q' to a_λ directly. The cost of this solution is at most $4 \cdot 2^{\ell_\lambda}$ (from the condition of the loop) plus $\delta(a_\lambda, q')$ (which is at most 2^{ℓ_λ} since $q' \in E_\lambda$). Overall, the cost of this solution is at most $5 \cdot 2^{\ell_\lambda}$; thus, the cost of the Steiner tree chosen by the algorithm is at most $10 \cdot 2^{\ell_\lambda}$, as it uses a 2-approximation. The cost of traversing this tree from q is thus at most $20 \cdot 2^{\ell_\lambda}$.

Second, the server possibly moves from its initial position a_λ to q (if λ is primary). The cost of this is at most $2^{\ell_\lambda - 1}$. Overall, the cost of a service λ is at most $O(1) \cdot 2^{\ell_\lambda}$. □

Lemma 3.8. ALG $\leq O(1) \cdot \left(\sum_{\lambda \in \Lambda^{\mathrm{p}}} 2^{\ell_\lambda} + \sum_{\lambda \in \Lambda^{\mathrm{c}}} 2^{\ell_\lambda}\right)$

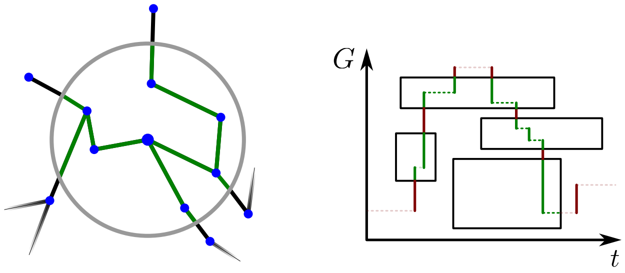
Proof. Using Proposition 3.7, we have that ALG $\leq O(1) \cdot \sum_{\lambda \in \Lambda} 2^{\ell_\lambda}$. Consider any non-primary service $\lambda \in \Lambda \backslash \Lambda^{\mathrm{p}}$. Since the service is non-primary, it certifies another service $\lambda' \in \Lambda^{\mathrm{c}}$, such that $\ell_{\lambda'} = \ell_\lambda - 4$. Thus, we have that $2^{\ell_\lambda} = 16 \cdot 2^{\ell_{\lambda'}}$; moreover, Proposition 3.6 implies that services are only certified once, and thus $\sum_{\lambda \in \Lambda \backslash \Lambda^{\mathrm{p}}} 2^{\ell_\lambda} \leq 16 \cdot \sum_{\lambda' \in \Lambda^{\mathrm{c}}} 2^{\ell_{\lambda'}}$. Overall, we have that

$$\mathrm{ALG} \leq O(1) \cdot \sum_{\lambda \in \Lambda} 2^{\ell_\lambda} \leq O(1) \cdot \left(\sum_{\lambda \in \Lambda^{\mathrm{p}}} 2^{\ell_\lambda} + \sum_{\lambda \in \Lambda^{\mathrm{c}}} 2^{\ell_\lambda}\right)$$

□

Charging Cylinders. Lemma 3.8 bounded the cost of the algorithm by the sum of two terms which correspond to primary and certified services. It remains to charge those terms to the optimal solution. To this end, we describe a method for charging costs to the optimal solution.

Charging balls. Recall that $B(v, r)$ denotes the ball of radius r centered at some point $v \in G$. Overloading notation, we use this



(a) (b)

Figure 1: Visualization of Intersections

terminology not only as a set of nodes, but also as a set of edges and "parts" of edges that exist within the ball; an informal visualization is given in Figure 1a. More formally, $B(v,r)$ contains all edges where both endpoints are in $B(v,r)$. In addition, when an edge e of weight w_e has exactly one endpoint u in $B(v,r)$, the part of e that belongs to $B(v,r)$ is the segment of weight $r - \delta(v,u)$ closest to u. It is easy to see that this definition preserves desirable properties for edges in a ball; in particular, note that the edges and parts of edges in $B(v_1, r_1)$ and in $B(v_2, r_2)$ are disjoint if $\delta(v_1, v_2) > r_1 + r_2$.

For the sake of charging costs, we create *cylinders.*

Definition 3.9 (cylinders). A cylinder is an ordered pair (B, I) where B is some shape in the metric space G, and I is a time interval.

As hinted by the word "cylinder", we later choose B to be a ball (or a perforated ball, defined later). The movement of the optimal solution inside the cylinder (i.e., during I and inside B) is then charged to.

Definition 3.10 (shape/cylinder charging). We use the following definitions to partition the costs of the optimal solution:

(1) Given a subgraph $G' \subseteq G$ and a shape B, we denote by $c(G' \cap B)$ the total weight of edges (or parts of edges) in G' that belong to B.
(2) Given a cylinder $\gamma = (B, I)$, define $c(\mathrm{OPT} \cap \gamma) := c\left(G^*_I \cap B\right)$, where G^*_I is the subgraph of edges traversed by OPT during I.

For a set Γ of cylinders, define $c(\mathrm{OPT} \cap \Gamma) := \sum_{\gamma \in \Gamma} c(\mathrm{OPT} \cap \gamma)$ for ease of notation. We now define disjointness for cylinders; a disjoint set of cylinders can charge to the optimal solution simultaneously, as they each charge to different movements of the server.

Definition 3.11 (disjoint cylinders). A pair of cylinders (B_1, I_1), (B_2, I_2) are called disjoint if either B_1, B_2 are disjoint, or I_1, I_2 are disjoint.

A set of cylinders whose metric shape is a ball can be seen in Figure 1b. Here, for the sake of visualization, we chose the metric space G to be a line. The cylinders thus appear as rectangles in the time-space plane. The tour of the optimal server over time appears as a line, in which the dotted segments represent the passage of time and the solid segments show movement through space. Only the length of solid segments inside a cylinder could be counted towards the intersection. Thus, since the cylinders in the figure are disjoint, the total charged amount does not exceed the total moving cost of the optimal solution. This is stated in Observation 1.

Observation 1. *Let Γ be a set of disjoint cylinders. Then*

$$c(\mathrm{OPT} \cap \Gamma) \le \mathrm{OPT}$$

With Observation 1, the way to use cylinders becomes clear: we want to construct a set of cylinders such that (a) their intersection with OPT is large, and (b) they are disjoint (or can be partitioned into a few disjoint subsets).

Bounding Primary Services. In this subsection, we focus on bounding the cost of primary services. For every service λ, define a^*_λ to be the final location of the optimum's server at t_λ. Define $\Lambda^{\mathrm{pf}} \subseteq \Lambda^{\mathrm{p}}$ to be the primary services λ such that $\delta\left(a^*_\lambda, q^\star_\lambda\right) \ge 2^{\ell_\lambda - 6}$. Proposition 3.12 shows that to bound the cost of primary services, it is enough to bound the cost of Λ^{pf}.

Proposition 3.12. $\sum_{\lambda \in \Lambda^{\mathrm{p}}} 2^{\ell_\lambda} \le O(1) \cdot \mathrm{OPT} + O(1) \cdot \sum_{\lambda \in \Lambda^{\mathrm{pf}}} 2^{\ell_\lambda}$.

Proof. Define the potential function $\phi(t) := 4\delta(a(t), a^*(t))$, where $a(t), a^*(t)$ are the server locations of the algorithm and the optimum at t, respectively. Note that the potential function equals 0 at the beginning of the input, and can only take on positive values. For a service $\lambda \in \Lambda^{\mathrm{p}}$, define Δ_λ to be the increase in potential function by the service λ through the movement of the algorithm's server in λ. Note that:

(1) The only server movements in the algorithm are in primary services (other services return the server to its previous location).
(2) Increases in potential due to movements in OPT sum to at most $4 \cdot \mathrm{OPT}$.

Therefore, we have the following:

$$\sum_{\lambda \in \Lambda^{\mathrm{p}}} \delta\left(q^\star_\lambda, a_\lambda\right) \le 4\mathrm{OPT} + \sum_{\lambda \in \Lambda^{\mathrm{p}}} \left(\delta\left(q^\star_\lambda, a_\lambda\right) + \Delta_\lambda\right) \tag{1}$$

Consider a service $\lambda \in \Lambda^{\mathrm{p}} \backslash \Lambda^{\mathrm{pf}}$, such that $\delta\left(a^*(t_\lambda), q^\star_\lambda\right) < 2^{\ell_\lambda - 6}$. In addition, note that the fact that λ is primary implies that $\ell_\lambda = \left\lceil \log \delta\left(a_\lambda, q^\star_\lambda\right) \right\rceil + 3$, and thus $\delta\left(a_\lambda, q^\star_\lambda\right) \ge 2^{\ell_\lambda - 4}$; we have

$$\begin{aligned} \Delta_\lambda &= 4 \cdot (\delta\left(a^*_\lambda, q^\star_\lambda\right) - \delta\left(a^*_\lambda, a_\lambda\right)) \\ &\le 4 \cdot \left(\delta\left(a^*_\lambda, q^\star_\lambda\right) - \delta\left(q^\star_\lambda, a_\lambda\right) + \delta\left(a^*_\lambda, q^\star_\lambda\right)\right) \\ &\le 4 \cdot (-2^{\ell_\lambda - 5}) = -2^{\ell_\lambda - 3} \end{aligned}$$

Observing that $\delta\left(a_\lambda, q^\star_\lambda\right) \le 2^{\ell_q - 3}$, we have $\delta\left(a_\lambda, q^\star_\lambda\right) + \Delta_\lambda \le 0$ for every $\lambda \in \Lambda^{\mathrm{p}} \backslash \Lambda^{\mathrm{pf}}$. Moreover, note that for every $\lambda \in \Lambda^{\mathrm{p}}$, it holds that $\Delta_\lambda \le 4\delta\left(a_\lambda, q^\star_\lambda\right)$, and thus $\delta\left(a_\lambda, q^\star_\lambda\right) + \Delta_\lambda \le 5\delta\left(a_\lambda, q^\star_\lambda\right)$.

Combining all observations, we get

$$\begin{aligned}
\sum_{\lambda\in\Lambda^{\mathrm{p}}} 2^{\ell_\lambda} &\leq 16 \sum_{\lambda\in\Lambda^{\mathrm{p}}} \delta\left(q^\star_\lambda, a_\lambda\right)\\
&\leq 64\cdot\mathrm{OPT} + 16\sum_{\lambda\in\Lambda^{\mathrm{p}}}\left(\delta\left(q^\star_\lambda, a_\lambda\right)+\Delta_\lambda\right)\\
&\leq 64\cdot\mathrm{OPT} + 16\sum_{\lambda\in\Lambda^{\mathrm{pf}}}\left(\delta\left(q^\star_\lambda, a_\lambda\right)+\Delta_\lambda\right)\\
&\leq 64\cdot\mathrm{OPT} + 80\sum_{\lambda\in\Lambda^{\mathrm{pf}}}\delta\left(q^\star_\lambda, a_\lambda\right)\\
&\leq 64\cdot\mathrm{OPT} + 10\sum_{\lambda\in\Lambda^{\mathrm{pf}}} 2^{\ell_\lambda}
\end{aligned}$$

where the first inequality is from $\delta\left(q^\star_\lambda, a_\lambda\right) \geq 2^{\ell_\lambda-4}$, the second inequality is through Equation (1), the third inequality is from the fact that for every $\lambda\in\Lambda^{\mathrm{p}}\setminus\Lambda^{\mathrm{pf}}$ we have $\delta\left(q^\star_\lambda, a_\lambda\right)+\Delta_\lambda \leq 0$, the fourth inequality is through $\delta\left(q^\star_\lambda, a_\lambda\right)+\Delta_\lambda \leq 5\delta\left(q^\star_\lambda, a_\lambda\right)$, and the final inequality is due to $\delta\left(q^\star_\lambda, a_\lambda\right) \leq 2^{\ell_\lambda-3}$. □

To finish bounding the cost of primary services, it is thus enough to prove the following lemma.

Lemma 3.13. $\sum_{\lambda\in\Lambda^{\mathrm{pf}}} 2^{\ell_\lambda} \leq O(\log\Delta)\cdot\mathrm{OPT}$.

Definition 3.14 (primary interals and cylinders). For every service $\lambda\in\Lambda^{\mathrm{pf}}$, we define:

1. The primary time interval $I_{\mathrm{p}}(\lambda) := (r_{q^\star_\lambda}, d_{q^\star_\lambda}]$.
2. The primary cylinder $\gamma_{\mathrm{p}}(\lambda) := \left(B\left(a_\lambda, 2^{\ell_\lambda-2}\right), I_{\mathrm{p}}(\lambda)\right)$.

We also define Γ^{p} to be the set of all primary cylinders of services from Λ^{pf}. In addition, for every i we define Γ^{p}_i to be the set of primary cylinders of level-i services from Λ^{pf}.

Proposition 3.15. *For every $\lambda\in\Lambda^{\mathrm{pf}}$, it holds that*

$$c\left(\mathrm{OPT}\cap\gamma_{\mathrm{p}}(\lambda)\right) \geq 2^{\ell_\lambda-6}$$

Proof. Since $\lambda\in\Lambda^{\mathrm{pf}}$, we know that the server of the optimal solution was at $q^\star_\lambda$ somewhere during $I_{\mathrm{p}}(\lambda)$, but was outside $B\left(q^\star_\lambda, 2^{\ell_\lambda-6}\right)$ at t_λ; thus, the optimal solution incurred a cost of at least $2^{\ell_\lambda-6}$ inside $B\left(q^\star_\lambda, 2^{\ell_\lambda-6}\right)$ during $I_{\mathrm{p}}(\lambda)$. Observing that $B\left(q^\star_\lambda, 2^{\ell_\lambda-6}\right) \subseteq B\left(a_\lambda, 2^{\ell_\lambda-2}\right)$ implies that $c\left(\mathrm{OPT}\cap\gamma_{\mathrm{p}}(\lambda)\right) \geq 2^{\ell_\lambda-6}$. □

Proposition 3.16. *For every i, Γ^{p}_i is a set of disjoint cylinders.*

Proof. Assuming otherwise, there exist i and two level-i services $\lambda_1,\lambda_2\in\Lambda^{\mathrm{pf}}$ such that $\gamma_{\mathrm{p}}(\lambda_1),\gamma_{\mathrm{p}}(\lambda_2)\in\Gamma^{\mathrm{p}}_i$ are not disjoint. This implies that $I_{\mathrm{p}}(\lambda_1)\cap I_{\mathrm{p}}(\lambda_2)\neq\emptyset$; hence, WLOG, assume that $t_{\lambda_1}\in I_{\mathrm{p}}(\lambda_2) = (r_{q^\star_{\lambda_2}}, d_{q^\star_{\lambda_2}}]$. Thus, $q^\star_{\lambda_2}$ was pending during λ_1, and moreover had level at most ℓ_{λ_2}. But since $\gamma_{\mathrm{p}}(\lambda_1),\gamma_{\mathrm{p}}(\lambda_2)$ are not disjoint, we have $\delta\left(a_{\lambda_1}, a_{\lambda_2}\right)\leq 2^{\ell_{\lambda_1}-1}$, but this implies that

$$\delta\left(a_{\lambda_1}, q^\star_{\lambda_2}\right) \leq \delta\left(a_{\lambda_1}, a_{\lambda_2}\right) + \delta\left(a_{\lambda_2}, q^\star_{\lambda_2}\right) \leq 2^{\ell_{\lambda_1}-1} + 2^{\ell_{\lambda_1}-3} \leq 2^{\ell_{\lambda_1}}$$

Thus, $q^\star_{\lambda_2}\in E_{\lambda_1}$. But requests in E_{λ_1} are either served by λ_1 or have their level increased to $\ell_{\lambda_1}+1$, in contradiction to λ_2 being primary. □

Proof of Lemma 3.13. The following holds:

$$\begin{aligned}
\sum_{\lambda\in\Lambda^{\mathrm{pf}}} 2^{\ell_\lambda} &\leq O(1)\cdot\sum_{\text{level } i} c\left(\mathrm{OPT}\cap\Gamma^{\mathrm{p}}_i\right)\\
&\leq O(1)\cdot\sum_{\text{level } i}\mathrm{OPT}\\
&O(\log\Delta)\cdot\mathrm{OPT}
\end{aligned}$$

where the first inequality is due to Proposition 3.15, the second inequality is due to Proposition 3.16, and the third inequality is due to the fact that there are only $O(\log\Delta)$ possible classes for primary services. □

Bounding Certified Services. In this subsection, we focus on bounding the cost of certified services; specifically, we prove the following lemma.

Lemma 3.17. $\sum_{\lambda\in\Lambda^{\mathrm{c}}} 2^{\ell_\lambda} \leq O(\log(\Delta n))\cdot\mathrm{OPT}$.

Definition 3.18 (σ_λ and $I_{\mathrm{c}}(\lambda)$). Let $\lambda\in\Lambda^{\mathrm{c}}$ be a certified service. Let $\lambda'\in\Lambda^{\mathrm{c}}$ be the certified service with maximum $\tau_{\lambda'}$ subject to $\ell_{\lambda'} = \ell_\lambda$, $\tau_{\lambda'}\leq t_\lambda$ and $\delta(a_{\lambda'}, a_\lambda)\leq 6\cdot 2^{\ell_\lambda}$. We define:

1. The time $\sigma_\lambda := \tau_{\lambda'}$ if λ' exists (otherwise, define $\sigma_\lambda = -\infty$).
2. The time interval $I_{\mathrm{c}}(\lambda) := (\sigma_\lambda, \tau_\lambda]$; note that $t_\lambda\in I_{\mathrm{c}}(\lambda)$.

Proposition 3.19. *Let $\lambda_1,\lambda_2\in\Lambda^{\mathrm{c}}$ be such that $\ell_{\lambda_1} = \ell_{\lambda_2} = \ell$ and $\delta\left(a_{\lambda_1}, a_{\lambda_2}\right)\leq 6\cdot 2^\ell$. Assuming WLOG that $t_{\lambda_1} < t_{\lambda_2}$, and letting λ be the level-$(\ell_{\lambda_1}+4)$ service that made λ_1 certified, it holds that $t_\lambda\in(\tau_{\lambda_1}, t_{\lambda_2}]$. (In particular, $\tau_{\lambda_1} < t_{\lambda_2}$.)*

Proof. The two possible cases which contradict our proposition are that $t_\lambda\leq\tau_{\lambda_1}$ or that $t_\lambda > t_{\lambda_2}$. If $t_\lambda\leq\tau_{\lambda_1}$, consider the triggering request $q^\star_\lambda$: this request was a witness for λ_1, and thus in E_{λ_1}; however, λ_1 chose which requests from E_{λ_1} to serve according to earliest deadline, and managed to serve all requests in E_{λ_1} of deadline $\leq\tau_{\lambda_1}$ (by definition). The existence of $q^\star_{\lambda_1}$ as a pending request at t_λ is therefore a contradiction.

Otherwise, if $t_\lambda > t_{\lambda_2}$, consider the service λ' which certified λ_2; it must also be the case that $t_{\lambda'} > t_{\lambda_2}$. Suppose that $t_\lambda < t_{\lambda'}$. In this case, observe that all witnesses for λ_2 at t_λ are in E_λ; therefore, these witnesses would no longer be witnesses for λ_2 after λ, in contradiction to one of them triggering λ' and certifying λ_2. Similarly, if $t_\lambda > t_{\lambda'}$, the service λ' would leave no witnesses for λ_1 to trigger λ. We thus again reached a contradiction. □

Definition 3.20 (certified cylinders). For a certified service $\lambda\in\Lambda^{\mathrm{c}}$, define the certified cylinder $\gamma_{\mathrm{c}}(\lambda) := (B\left(q^\star_\lambda, 3\cdot 2^{\ell_q}\right), I_{\mathrm{c}}(\lambda))$. Define Γ^{c} to be the set of all certified cylinders; in addition, for every i define Γ^{c}_i to be the set of certified cylinders formed from level-i services.

Proposition 3.21. *For every i, the set Γ^{c}_i is a set of disjoint cylinders.*

Proof. Consider any two cylinders $\gamma_{\mathrm{c}}(\lambda_1),\gamma_{\mathrm{c}}(\lambda_2)\in\Gamma^{\mathrm{c}}_i$. If it holds that $\delta\left(a_{\lambda_1}, a_{\lambda_2}\right) > 6\cdot 2^i$, then the cylinders are spatially

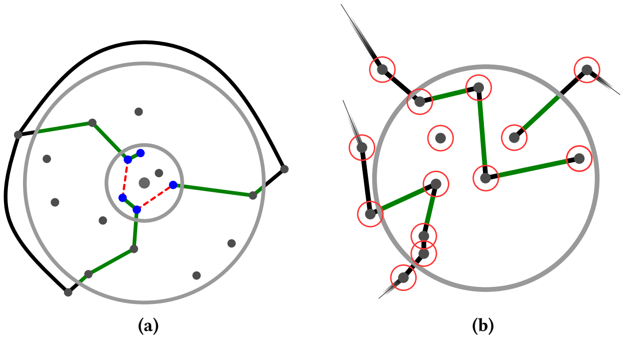


Figure 2: Visualizations from the analysis of Algorithm 1.

disjoint and we are done. Thus, assume that $\delta(a_{\lambda_1}, a_{\lambda_2}) \le 6 \cdot 2^i$, and WLOG assume that $t_{\lambda_1} < t_{\lambda_2}$. Proposition 3.19 implies that $\tau_{\lambda_1} < t_{\lambda_2}$; from the definition of σ_{λ_2}, it is thus also the case that $\sigma_{\lambda_2} \ge \tau_{\lambda_1}$. Thus, the intervals $I_{\mathrm{c}}(\lambda_1), I_{\mathrm{c}}(\lambda_2)$ are disjoint, and thus $\gamma_{\mathrm{c}}(\lambda_1), \gamma_{\mathrm{c}}(\lambda_2)$ are disjoint. □

Having defined the certified cylinders and shown a disjointness property in Proposition 3.21, we want to show that the optimal solution has a large intersection with these cylinders. We show this by claiming that the release-to-deadline intervals for requests in E_λ are contained in $I_{\mathrm{c}}(\lambda)$, and thus must be served by the optimal solution in this interval. Therefore, a Steiner tree spanning E_Q can be charged to the optimal solution in this time interval. However, one must still claim that enough of this cost takes place within the ball defining the cylinder. This is possible as the requests of E_λ are in $B(a_\lambda, 2^{\ell_\lambda})$, while the radius of $\gamma_{\mathrm{c}}(\lambda)$ is $3 \cdot 2^{\ell_\lambda}$.

Proposition 3.22. *Consider a set of points $V \subseteq B(\rho, r)$, and let G' be a subgraph of G that connects V. Then it holds that*

$$\mathrm{ST}^*(V) \le 2 \cdot c\big(G' \cap B(\rho, 3r)\big)$$

Proof. Consider the edges in G' contained in $B(\rho, 3r)$. If those edges connect all points in V, then they form a valid solution for Steiner tree on V, and thus we are done. Otherwise, these edges partition V into connected components $V_1, \cdots, V_k$. Since V are connected in G', these connected components must be connected somewhere outside $B(\rho, r)$; in particular, connected component contains a path which exits $B(\rho, 3r)$; since the connected component contains a point in $V \subseteq B(\rho, r)$, the total weight of this component is at least $2r$.

We can convert $G \cap B(\rho, 3r)$ into a Steiner tree solution for V by connecting at most $k-1$ pairs of points from V directly, such that the points of each pair belong to different components V_i, V_j. Since the diameter of V is at most $2r$, and the cost of each connected component is at least $2r$, this modification at most doubles the cost of $G' \cap B(\rho, 3r)$. This completes the proof. □

A visual description of the proof of Proposition 3.22 is given in Figure 2a. Here, the terminals V (in blue) are connected by the subgraph G' such that the restriction to $B(\rho, 3r)$ creates three connected components. As the terminals are all in $B(\rho, r)$, adding the two red, dashed edges would augment this restriction to a Steiner tree solution for V, where each edge costs at most the diameter of the smaller ball (i.e., $2r$).

Proposition 3.23. *For every certified service $\lambda \in \Lambda^{\mathrm{c}}$ and request $q \in Q_\lambda$, it holds that $(r_q, d_q] \subseteq (\sigma_\lambda, \tau_\lambda]$.*

Proof. Define $\ell = \ell_q$. By definition, it holds that $d_q \le \tau_\lambda$. It remains to show that $r_q \ge \sigma_\lambda$. If $\sigma_\lambda = -\infty$, we are done; otherwise, by the definition of σ_λ, there exists a certified service λ' such that $\ell_{\lambda'} = \ell$, $\sigma_\lambda = \tau_{\lambda'}$ and $\delta(a_{\lambda'}, a_\lambda) \le 6 \cdot 2^{\ell_\lambda}$. Applying Proposition 3.19, there exists a level-$(\ell+4)$ service λ'' in $(\tau_{\lambda'}, t_\lambda]$, such that $\delta\left(q^\star_{\lambda''}, a_{\lambda'}\right) \le 2^\ell$.

Now, observe that

$$\begin{aligned}\delta(q, a_{\lambda''}) &\le \delta(q, a_\lambda) + \delta(a_\lambda, a_{\lambda'}) + \delta\left(a_{\lambda'}, q^\star_{\lambda''}\right) + \delta\left(q^\star_{\lambda''}, a_{\lambda''}\right) \\ &\le 2^{\ell_\lambda} + 6 \cdot 2^{\ell_\lambda} + 2^{\ell_\lambda} + 2^{\ell_\lambda + 1} = 10 \cdot 2^{\ell_\lambda} \le 2^{\ell_{\lambda''}} \qquad (2)\end{aligned}$$

Assume for contradiction that $r_q < \sigma_\lambda$, which implies that $r_q < \tau_{\lambda'}$. Since $q \in Q_\lambda$, we have that q was pending during $(\tau_{\lambda'}, t_\lambda]$. Thus, q was pending during λ''; combining with Equation (2), it must be that $q \in E_{\lambda''}$. But then it must be that λ'' raised the level of q to $\ell_{\lambda''} + 1 = \ell_\lambda + 5$, in contradiction to $q \in Q_\lambda \subseteq E_\lambda$. □

Proposition 3.24. *For every certified cylinder $\gamma_{\mathrm{c}}(\lambda)$, it holds that $2^{\ell_\lambda - 1} \le c(\mathrm{OPT} \cap \gamma_{\mathrm{c}}(\lambda))$.*

Proof. Consider the union of edges traversed by the optimum during $(\sigma_\lambda, \tau_\lambda]$, and denote it by G^*; note that $c(\mathrm{OPT} \cap \gamma_{\mathrm{c}}(\lambda)) = c\big(G^* \cap B\big(a_\lambda, 3 \cdot 2^{\ell_\lambda}\big)\big)$. From Proposition 3.23, G^* must connect Q_λ; Noting that $Q_\lambda \subseteq B(a_\lambda, 2^{\ell_\lambda})$, Proposition 3.22 implies $\mathrm{ST}^*(Q_\lambda) \le 2 \cdot c\left(G^* \cap B\left(q^\star_\lambda, 3 \cdot 2^{\ell_\lambda}\right)\right)$. Now, note that since λ is a certified service, Line 15 was reached, and thus the approximate solution of the algorithm for $\mathrm{ST}(Q_\lambda \cup \{a_\lambda\})$ had cost at least $4 \cdot 2^{\ell_\lambda}$. Since the Steiner-tree algorithm is a 2-approximation, we have $\mathrm{ST}^*(Q_\lambda \cup \{a_\lambda\}) \ge 2 \cdot 2^{\ell_\lambda}$. Finally, since a_λ can be connected directly to any request in Q_λ at cost at most 2^{ℓ_λ}, we have $\mathrm{ST}^*(Q_\lambda \cup \{a_\lambda\}) \le \mathrm{ST}^*(Q_\lambda) + 2^{\ell_\lambda}$, which therefore implies $\mathrm{ST}^*(Q_\lambda) \ge 2^{\ell_\lambda}$. Combining, we have that $2^{\ell_\lambda - 1} \le c(\mathrm{OPT} \cap \gamma_{\mathrm{c}}(\lambda))$. □

Proof of Lemma 3.17. The following holds:

$$\begin{aligned}\sum_{\lambda \in \Lambda^{\mathrm{c}}} 2^{\ell_\lambda} &\le O(1) \cdot \sum_{\text{level } i} c(\Gamma_i^{\mathrm{c}} \cap \mathrm{OPT}) \\ &\le O(1) \cdot \sum_{\text{level } i} \mathrm{OPT} \\ &\le O(\log(\Delta n)) \cdot \mathrm{OPT}\end{aligned}$$

where the first inequality is due to Proposition 3.24, the second inequality is due to Proposition 3.21, and the final inequality is due to the fact that the number of possible classes for services is $O(\log(\Delta n))$. □

Proof of Theorem 3.2. We have

$$\begin{aligned}\mathrm{ALG} &\le O(1) \cdot \left(\sum_{\lambda \in \Lambda^{\mathrm{c}}} 2^{\ell_\lambda} + \sum_{\lambda \in \Lambda^{\mathrm{pf}}} 2^{\ell_\lambda} + \mathrm{OPT}\right) \\ &\le O(\log(\Delta n)) \cdot \mathrm{OPT}\end{aligned}$$

where the first inequality is due to Lemma 3.8 and Proposition 3.12, and the second inequality is due to Lemmas 3.13 and 3.17. □

Improved Analysis through Perforated Cylinders. We now alter our cylinder construction to obtain Theorem 3.1. We do so by changing the shape of cylinders in the metric space from a ball to a *perforated ball.* A perforated ball used in our proofs is formed from a ball of radius r by removing balls of radius $\frac{r}{c \cdot n^2}$ around every point in the metric space, for some constant $c > 1$. Since the radii of removed balls are small, we claim that the intersection of cylinders using these new, perforated balls with OPT is only smaller by a constant factor from the original intersection. However, the perforation ensures that cylinders with an $\Omega(\log n)$ gap do not intersect, yielding increased disjointness and better competitiveness. An informal visualization of perforated balls appears in Figure 2b, which shows the intersection of a subgraph with such a ball. A formal description, as well as the proofs of Lemmas 3.25 and 3.26 appear in Appendix B.

Lemma 3.25 (improved Lemma 3.13).

$$\sum_{\lambda \in \Lambda^{\mathrm{pf}}} 2^{\ell_\lambda} \leq O(\log n) \cdot \mathrm{OPT}.$$

Lemma 3.26 (improved Lemma 3.17).

$$\sum_{\lambda \in \Lambda^{\mathrm{c}}} 2^{\ell_\lambda} \leq O(\log n) \cdot \mathrm{OPT}.$$

Proof of Theorem 3.1. The proof of the theorem is identical to that of Theorem 3.2, except that Lemmas 3.13 and 3.17 are replaced with Lemmas 3.25 and 3.26. □

4 CONCLUSIONS AND FUTURE DIRECTIONS

In this paper, we introduced the first deterministic algorithms for online service with deadlines/delay for general metric spaces (of subpolynomial competitiveness). Our algorithms also improve upon the best known randomized algorithms for the problem. While superconstant lower bounds for this problem are not yet known, there is evidence that the $O(\log n)$ competitiveness shown in this paper is tight. This is suggested by the fact that previous to our work, $O(\log n)$-competitiveness was the best known bound for both multilevel aggregation (see [3]) and online service with delay on an equidistant line (see [7]), which are both special cases of the problem considered in this paper. Introducing a superconstant lower bound for this problem would thus be a great future direction. Also note that currently, no separation is known between deterministic and randomized algorithms: for service with deadlines/delay, as well as for the mentioned special cases, the best known algorithms are deterministic, and there exists no known superconstant lower bound even for deterministic algorithms.

In addition, we believe that our techniques could be extended to multiple servers, yielding a deterministic algorithm for k-server with delay. This would require some combination of our techniques with existing deterministic techniques for k-server on general metric spaces, e.g., the work function algorithm [20].

A ONLINE SERVICE WITH DELAY

This section considers the online service with delay problem. For this problem, we prove the following theorem.

Theorem A.1. *There exists a $O(\log n)$-competitive deterministic algorithm for online service with delay.*

A.1 The Algorithm

Prize-Collecting Steiner tree. Similar to the algorithm for deadlines, the algorithm for delay also requires an approximation algorithm for Steiner tree. However, we now consider the prize-collecting variant of the Steiner tree problem. In this variant, we are given a set of terminals and a root node; in addition, each terminal has an associated *penalty.* A solution is a subgraph that connects some subset of the terminals to the root node. The cost of that solution is the total weight of the subgraph, plus the penalties for terminals that were not connected to the root node.

There exists a 3-approximation for prize-collecting Steiner tree, due to Hajiaghayi et al. [17], which we denote PCST. We use $\mathrm{PCST}(U, \pi; r)$ to denote running the approximation over the input graph G, with terminal set U, penalty function π (that maps from terminal to its penalty), and root node r. As before, we use $\mathrm{PCST}(U, \pi; r)$ to refer to the cost of the approximate solution (edge and penalty cost). We also use $\mathrm{PCST}^*(U, \pi; r)$ to refer to the optimal solution for the same input (and its cost).

Algorithm's description. As in the deadline algorithm, every pending request q has a level ℓ_q (which is initially $-\infty$), and we define the adjusted level of q to be $\bar{\ell}_q := \max\{\ell_q, \lceil \log \delta(q, a) \rceil\}$, where a is the current location of the server. In addition, for every request q the algorithm maintains an investment counter h_q (which is initially 0). This counter is raised by a service to pay for (past or future) delay of a request. We denote by $\ell_q(t), \bar{\ell}_q(t), h_q(t)$ the values of $\ell_q, \bar{\ell}_q, h_q$ at time t (if a service takes place at t, this refers to the values immediately before the service). We also define the *residual delay* of q at t to be $y_q(t) := \left(d_q(t) - h_q(t)\right)^+$; intuitively, this is the amount of current delay which no service has paid for.

The following definition of a critical level is used to trigger services in the algorithm.

Definition A.2 (critical level). Fix any time t, and a level ℓ.

(1) Define $Y_\ell(t)$ to be the total residual delay of requests q s.t. $\bar{\ell}_q \leq \ell$.
(2) We say that ℓ is *critical* if $Y_\ell(t) \geq 2^\ell$.

The pseudocode for the algorithm is given in Algorithm 2. Whenever a level ℓ becomes critical, the function UponCritical is called, which initiates a service λ; the level of this service ℓ_λ is set to be $\ell + 3$. The service identifies the triggering set of requests $Q^\star_\lambda$, which are the requests whose total residual delay became critical (that is, requests of adjusted level at most $\ell_\lambda - 3 = \ell$ with positive residual delay). If there exists no triggering request of level at least $\ell_\lambda - 4$ (i.e., at least $\ell - 1$), the service is called primary.

For a primary service λ, the algorithm attempts to identify a new location for placing the server after the service concludes. If a constant fraction of the residual delay of $Q^\star_\lambda$ exists inside a small radius ball, the center of that ball will be the new location of the server. Otherwise, the final location of the server will be its starting location.

A service λ identifies all requests eligible to the service, which are all requests with adjusted level at most ℓ_λ; these are denoted E_λ. The service then resets the residual delay of all requests in E_λ. (Note, in particular, that this resets the residual delay of all triggering requests in $Q^\star_\lambda$.) Then, the service performs time forwarding: it observes the future delay of requests in E_λ, and attempts to "pay" for those requests until a point in time furthest in the future. "Paying" for a request q until time τ can be done either through serving that request, or by increasing its investment counter to at least $d_q(\tau)$. Choosing between these options is done through calling PCST on the eligible requests with a penalty function which represents the required increase to investment counters. Specifically, the algorithm finds the first time τ in which the cost of PCST exceeds $c \cdot 2^{\ell_\lambda}$, for some constant c; the algorithm will serve requests and increase counters according to the PCST solution for time τ.

Finally, at the end of the service, eligible requests that are still pending are upgraded to a level higher than that of the service, and the server moves to its new final location (if applicable).

Algorithm 2: Online Service with Delay

```
1  Event Function UponRequest($q$)
2      set $\ell_q \leftarrow -\infty$, $h_q \leftarrow 0$.
3  Event Function UponCritical($\ell$)
4      start a new service, denoted by $\lambda$, and set $\ell_\lambda \leftarrow \ell + 3$.
5      let $t$ be the time, $a$ be the server's location, and $Q'$ be the set of pending requests.
6      let $Q^\star_\lambda \leftarrow \{q' \in Q' | \overline{\ell}_q \le \ell_\lambda - 3 \wedge y_q(t) > 0\}$.
7      if for every $q' \in Q^\star_\lambda$ we have $\ell_{q'} < \ell_\lambda - 4$ then say $\lambda$ is primary else $\lambda$ is not primary.
8      if $\lambda$ is primary and there exists $a' \in G$ s.t. $y_R(t) > 2^{\ell_\lambda - 4}$ where $R := Q^\star_\lambda \cap B(a', 2^{\ell_\lambda - 8})$ then
9          define $a'$ as mentioned.
10     else
11         set $a' \leftarrow$ Null.
12     set $E_\lambda \leftarrow \{q' \in Q' | \overline{\ell}_q \le \ell_\lambda\}$.
13     foreach $q' \in E_\lambda$ do set $h_{q'} \leftarrow \max\{h_{q'}, d_{q'}(t)\}$.
         // Zero residual delay.
14     set $Q_\lambda \leftarrow \{q\}, S \leftarrow \emptyset$.
15     for every time $t' \ge t$, define the penalty function
         $\pi_{t'} : E_\lambda \longrightarrow \mathbb{R}^+ \cup \{0\}$ such that
         $\pi_{t'}(q') = (d_{q'}(t') - h_{q'})^+$ for every $q' \in E_\lambda$.
16     let $\tau \ge t$ be the first time in which
         $\mathrm{PCST}(E_\lambda, \pi_\tau; a) \ge 6 \cdot 2^{\ell_\lambda}$.
17     let $S$ to be the solution $\mathrm{PCST}(E_\lambda, \pi_\tau; a)$, and let $Q_\lambda \subseteq E_\lambda$ be the set of requests served by $S$.
18     perform DFS tour of $S$, serving $Q_\lambda$ and finishing at $a$.
19     foreach $q' \in E_\lambda \backslash Q_\lambda$ do
20         set $h_{q'} \leftarrow \max\{h_{q'}, d_{q'}(\tau)\}$.
21         set $\ell_{q'} \leftarrow \ell_\lambda + 1$.
22     if $a' \ne Null$ then move the server from $a$ to $a'$.
```

A level-ℓ service is triggered when level $\ell - 3$ becomes critical, i.e., when requests of adjusted level at most $\ell - 3$ gather large residual delay. Figure 3 gives some intuition about how the residual delay of those triggering requests are distributed. In Figure 3, the delay of the triggering requests of a service λ is shown as a heatmap inside the ball $B(a_\lambda, 2^{\ell_\lambda - 3})$ (more delay is a deeper shade of red). Figure 3a shows a pattern that can only belong to a nonprimary service: in primary services, the outer ring $B(a_\lambda, 2^{\ell_\lambda - 3}) \setminus B(a_\lambda, 2^{\ell_\lambda - 5})$ must contain a constant fraction of the residual delay (as shown in the proof of Proposition A.7). In secondary services, the server's final location is the same as its initial location. Figure 3b shows a delay pattern which might belong to a primary service. If this is the case, the server will again finish at its initial location, as delay is not concentrated in a low-radius ball (a' is not defined in Line 11). Figure 3c shows a delay pattern which is highly concentrated in a low-radius ball. Thus, if this pattern belongs to a primary service, the server would move to the center of the low-radius ball at the end of the service.

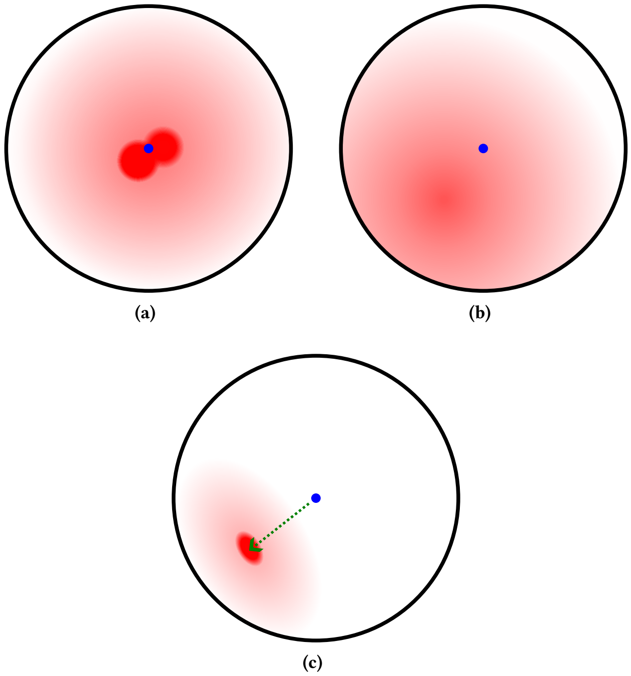


Figure 3: Possible distributions for residual delay of triggering requests.

A.2 Analysis

We now focus on proving Theorem A.1.

Definition A.3 (basic service definitions). Let λ be a service. We define:

- The *triggering requests* $Q^\star_\lambda$ to be as defined in Line 6.
- The location a_λ to be the initial location of the server when λ is triggered.
- The term t_λ to be the time of service λ.
- The request set Q_λ to be the requests served by λ (i.e., the final value of the variable of that name in UponDeadline).
- The request set E_λ as defined in Line 12; these requests are called *eligible for* λ.
- The *forwarding time* of λ, denoted τ_λ, to be τ as defined in Line 16.

Definition A.4 (witness requests and certified services). At a certain point in time, a request q is called a *witness* for a service λ if its level ℓ_q was last assigned to by λ (at Line 18).

For every non-primary service λ', let $q \in Q^\star_{\lambda'}$ be an arbitrary request such that $\ell_q(t_\lambda) \ge \ell_{\lambda'} - 4$, and let λ be the service for which q is a witness. We say that λ' *certifies* λ, and call λ a *certified* service. (Note that the chosen request q is unique, such that every service certifies at most one other service.)

Proposition A.5. *If λ' certifies λ, then $\ell_\lambda + 4 \le \ell_{\lambda'} \le \ell_\lambda + 5$.*

Proof. Consider the request $q \in Q^\star_{\lambda'}$ which was a witness for λ. It holds that $\ell_q(t_{\lambda'}) = \ell_\lambda + 1$. In addition, it holds that $\ell_q(t_{\lambda'}) \le \ell_{\lambda'} - 3$ (as $q \in Q^\star_{\lambda'}$) and that $\ell_q(t_{\lambda'}) \ge \ell_{\lambda'} - 4$ (as q made λ' certify λ). □

Proposition A.6. *Throughout the algorithm, for every level i it holds that $Y_i \le 2^{i+1}$ at every point in time.*

Proof. Continuous delay growth cannot break this proposition, as the algorithm triggers a service whenever a level i becomes critical (i.e., when Y_i reaches 2^i), and this service then zeroes Y_i. The only conceivable way for this to happen is upon moving the server from a to a' in Line 22; indeed, this changes the adjusted levels of requests, possibly increasing $Y_i(t)$ past 2^i. However, we claim that this cannot increase Y_ℓ too much.

The server moved from a to a' during some service λ, triggered by some level ℓ becoming critical; it holds that $\ell_\lambda = \ell + 3$. Let t^-, t^+ be the times immediately before and after Line 22 in λ. At t^-, there are no requests with adjusted level less than $\ell + 4$, as ensured by *Line* 21; in particular, $Y_i = 0$ for every $i \le \ell + 3$. Moreover, it holds that $Y_i \le 2^i$ for every $i > \ell + 3$, as only the maximal critical level triggers a service. Thus, $Y_i(t^-) \le 2^i$ for all i.

Consider any pending request q at t^-; as mentioned, $\bar\ell_q(t^-) \ge \ell + 4$. We claim that $\bar\ell_q(t^+) \ge \bar\ell_q(t^-) - 1$; if this claim is correct, then for every i we have $Y_i(t^+) \le Y_{i+1}(t^-) \le 2^{i+1}$, and the proof is complete.

If $\bar\ell_q(t^-) = \ell_q(t^-)$, the claim holds as levels do not decrease. Otherwise, we have that $\delta(q, a) > 2^{\bar\ell_q(t^-)-1}$. But then the triangle inequality implies that $\delta(q, a') \ge \delta(q, a) - \delta(a, a') > 2^{\bar\ell_q(t^-)-1} - 2^{\ell+1} \ge 2^{\bar\ell_q(t^-)-2}$, where the final inequality uses $\bar\ell_q(t^-) \ge \ell + 4$. This implies that $\bar\ell_q(t^+) \ge \bar\ell_q(t^-) - 1$, completing the proof. □

Proposition A.7. *During a service λ, if the algorithm moves its server from a to a' in Line 22, then $2^{\ell_\lambda - 5} - 2^{\ell_\lambda - 8} \le \delta(a, a') \le 2^{\ell_\lambda - 3} + 2^{\ell_\lambda - 8}$.*

Proof. First, we prove that $\delta(a, a') \ge 2^{\ell_\lambda - 5} - 2^{\ell_\lambda - 8}$. Since λ was started, we know that $Y_{\ell_\lambda - 3} \ge 2^{\ell_\lambda - 3}$. From Proposition A.6, we know that $Y_{\ell_\lambda - 5} \le 2^{\ell_\lambda - 4}$; moreover, the service λ is primary, and thus $Q^\star_\lambda$ contains requests of level at most $\ell_\lambda - 5$. Thus, the total residual delay of $Q^\star_\lambda$ incurred inside $B(a_\lambda, 2^{\ell_\lambda - 5})$ is at most $2^{\ell_\lambda - 4}$. Now note that if $\delta(a, a') < 2^{\ell_\lambda - 5} - 2^{\ell_\lambda - 8}$ then $B(a', 2^{\ell_\lambda - 8}) \subseteq B(a, 2^{\ell_\lambda - 5})$, which contradicts the definition of a'.

Second, we prove that $\delta(a, a') \le 2^{\ell_\lambda - 3} + 2^{\ell_\lambda - 8}$. Assuming otherwise that $\delta(a, a') > 2^{\ell_\lambda - 3} + 2^{\ell_\lambda - 8}$, we have that $B(a', 2^{\ell_\lambda - 8})$ and $B(a, 2^{\ell_\lambda - 3})$ are disjoint, in contradiction to the former containing much of the residual delay of λ. □

We define the cost of a service λ, denoted $c(\lambda)$, to be the total movement of the algorithm's server in λ plus the total amount by which investment counters are raised in λ.

Proposition A.8. ALG $\le \sum_{\lambda \in \Lambda} c(\lambda)$.

Proof. Note that every request q is eventually served, and upon service h_q is at least the delay cost of that request. Thus, the sum of counters upper-bounds delay costs, and the raising of every counter is counted towards $c(\lambda)$ for some λ. All movement costs in ALG are also attributed to the cost of some service. □

Proposition A.9. *The total cost of service λ is $O(1) \cdot 2^{\ell_\lambda}$.*

Proof. From Proposition A.6, it holds that $Y_{\ell_\lambda} \le 2^{\ell_\lambda + 1}$; thus, $2^{\ell_\lambda + 1}$ bounds the cost of raising counters on Line 13.

In addition, the cost of traversing the PCST solution on Line 18 and investing in counters in Line 20 can be bounded using the following argument: in the previous iteration, the cost of the PCST solution was less than $6 \cdot 2^{\ell_\lambda}$, from the condition of the loop, which implies that the cost of the optimal solution was less than $6 \cdot 2^{\ell_\lambda}$; however, delay rises continuously, and thus this optimal solution applies to the final iteration as well (at cost at most $6 \cdot 2^{\ell_\lambda}$). Since the approximation algorithm PCST that we use is a 3-approximation, its cost of its output can be bounded by $18 \cdot 2^{\ell_\lambda}$. The penalty part of the solution is paid exactly in Line 20, while the served part of the solution is traversed in DFS (at double the cost). Thus, the total cost of Line 18 and Line 20 is at most $36 \cdot 2^{\ell_\lambda}$.

Finally, the cost of moving the server in Line 22 (if it takes place) is at most $2^{\ell_\lambda - 3} + 2^{\ell_\lambda - 5}$. □

Proposition A.10. ALG $\le O(1) \cdot \left(\sum_{\lambda \in \Lambda^{\mathrm{p}}} 2^{\ell_\lambda} + \sum_{\lambda \in \Lambda^{\mathrm{c}}} 2^{\ell_\lambda}\right)$.

Proof. The proof is similar to that of Lemma 3.8 for the deadline case. Every non-primary service of level ℓ certifies another service of level at least $\ell - 5$ (through Proposition A.5). Since every service is certified at most once, it holds that $\sum_{\lambda \in \Lambda \setminus \Lambda^{\mathrm{p}}} 2^{\ell_\lambda} \le O(1) \cdot \sum_{\lambda \in \Lambda^{\mathrm{c}}} 2^{\ell_\lambda}$. □

Bounding Primary Services. As in the argument for deadlines, we identify a subset of primary services which we need to bound. In fact, for delay, we identify two such subsets. Define a^*_λ to be the final location of the optimum's server at t_λ. We define two disjoint subsets of the primary services Λ^{p}:

1. Services in which the algorithm's server ended at the starting location (i.e. Line 22 did not run), denoted Λ^{ps}.
2. Services in which the algorithm's server moved to some location a' (i.e. Line 22 ran) such that $\delta\left(a^*_\lambda, a'\right) \ge 2^{\ell_\lambda - 7}$, denoted Λ^{pf}.

As stated in Proposition A.11, to bound the cost of all primary services it is enough to bound the cost of these two subsets.

Proposition A.11.

$$\sum_{\lambda \in \Lambda^{\mathrm{p}}} 2^{\ell_\lambda} \le O(1) \cdot \mathrm{OPT} + O(1) \cdot \sum_{\lambda \in \Lambda^{\mathrm{pf}}} 2^{\ell_\lambda} + O(1) \cdot \sum_{\lambda \in \Lambda^{\mathrm{ps}}} 2^{\ell_\lambda}$$

Proof. The proof is similar to the proof of Proposition 3.12. We define the potential function $\phi(t) := 4\delta(a(t), a^*(t))$, where $a(t), a^*(t)$ at the locations of the algorithm's server and the optimum's server at t, respectively. Note that the potential function equals 0 at the beginning of the input, and can only take on positive values. Note that in Λ^{ps}, the final and initial server locations are the same; we thus only consider services in $\Lambda^{\mathrm{p}} \setminus \Lambda^{\mathrm{ps}}$ in the potential argument. For every service $\lambda \in \Lambda^{\mathrm{p}} \setminus \Lambda^{\mathrm{ps}}$, we define a'_λ to be the final location of the server in λ (the value of a' in UponCritical). Following the argument for Proposition 3.12, we have

$$\sum_{\lambda \in \Lambda^{\mathrm{p}} \setminus \Lambda^{\mathrm{ps}}} \delta\left(a'_\lambda, a_\lambda\right) \leq 4\mathrm{OPT} + \sum_{\lambda \in \Lambda^{\mathrm{p}} \setminus \Lambda^{\mathrm{ps}}} \left(\delta\left(a'_\lambda, a_\lambda\right) + \Delta_\lambda\right) \qquad (3)$$

Now, consider a service $\lambda \in \Lambda^{\mathrm{p}} \setminus (\Lambda^{\mathrm{pf}} \cup \Lambda^{\mathrm{ps}})$. For ease, define $a^*_\lambda := a^*(t_\lambda)$. After λ the algorithm moves its server to a'_λ such that $\delta\left(a'_\lambda, a^*_\lambda\right) \leq 2^{\ell_\lambda - 7}$. In addition, Proposition A.7 implies that $\delta\left(a_\lambda, a'_\lambda\right) \geq 2^{\ell_\lambda - 5} - 2^{\ell_\lambda - 8} = 7 \cdot 2^{\ell_\lambda - 8}$. Therefore:

$$\begin{aligned}
\Delta_\lambda &\leq 4(\delta\left(a'_\lambda, a^*_\lambda\right) - \delta\left(a_\lambda, a^*_\lambda\right)) \\
&\leq 4 \cdot \left(\delta\left(a'_\lambda, a^*_\lambda\right) - \delta\left(a_\lambda, a'_\lambda\right) + \delta\left(a'_\lambda, a^*_\lambda\right)\right) \\
&\leq 4 \cdot \left(2^{\ell_\lambda - 6} - \delta\left(a_\lambda, a'_\lambda\right)\right) \\
&\leq 4 \cdot (-\frac{3}{7} \cdot \delta\left(a_\lambda, a'_\lambda\right)) \\
&\leq -\delta\left(a_\lambda, a'_\lambda\right)
\end{aligned}$$

where the second inequality is due to the triangle inequality. Therefore we have $\delta\left(a_\lambda, a'_\lambda\right) + \Delta_\lambda \leq 0$ for every $\lambda \in \Lambda^{\mathrm{p}} \setminus (\Lambda^{\mathrm{pf}} \cup \Lambda^{\mathrm{ps}})$. Moreover, note that for every $\lambda \in \Lambda^{\mathrm{p}} \setminus \Lambda^{\mathrm{ps}}$ we have $\Delta_\lambda \leq 4\delta\left(a_\lambda, a'_\lambda\right)$. Finally, note that for a service $\lambda \in \Lambda^{\mathrm{p}} \setminus \Lambda^{\mathrm{ps}}$, Proposition A.7 implies that $\delta\left(a_\lambda, a'_\lambda\right) \geq 2^{\ell_\lambda - 5} - 2^{\ell_\lambda - 8} = 7 \cdot 2^{\ell_\lambda - 8}$, and thus $2^{\ell_\lambda} \leq \frac{256}{7} \cdot \delta\left(a_\lambda, a'_\lambda\right)$. In addition, $\delta\left(a_\lambda, a'_\lambda\right) \leq 2^{\ell_\lambda - 3} + 2^{\ell_\lambda - 8} = \frac{33}{256} \cdot 2^{\ell_\lambda}$. Combining all observations, we get

$$\begin{aligned}
\sum_{\lambda \in \Lambda^{\mathrm{p}}} 2^{\ell_\lambda} &\leq \sum_{\lambda \in \Lambda^{\mathrm{ps}}} 2^{\ell_\lambda} + \sum_{\lambda \in \Lambda^{\mathrm{p}} \setminus \Lambda^{\mathrm{ps}}} 2^{\ell_\lambda} \\
&\leq \sum_{\lambda \in \Lambda^{\mathrm{ps}}} 2^{\ell_\lambda} + \frac{256}{7} \cdot \sum_{\lambda \in \Lambda^{\mathrm{p}} \setminus \Lambda^{\mathrm{ps}}} \delta\left(a_\lambda, a'_\lambda\right) \\
&\leq \sum_{\lambda \in \Lambda^{\mathrm{ps}}} 2^{\ell_\lambda} + \frac{256}{7} \cdot \left(4\mathrm{OPT} + \sum_{\lambda \in \Lambda^{\mathrm{p}} \setminus \Lambda^{\mathrm{ps}}} (\delta\left(a_\lambda, a'_\lambda\right) + \Delta_\lambda)\right) \\
&\leq \sum_{\lambda \in \Lambda^{\mathrm{ps}}} 2^{\ell_\lambda} + \frac{256}{7} \cdot \left(4\mathrm{OPT} + \sum_{\lambda \in \Lambda^{\mathrm{pf}}} (\delta\left(a_\lambda, a'_\lambda\right) + \Delta_\lambda)\right) \\
&\leq \sum_{\lambda \in \Lambda^{\mathrm{ps}}} 2^{\ell_\lambda} + \frac{256}{7} \cdot \left(4\mathrm{OPT} + 5\sum_{\lambda \in \Lambda^{\mathrm{pf}}} \delta\left(a_\lambda, a'_\lambda\right)\right) \\
&\leq \sum_{\lambda \in \Lambda^{\mathrm{ps}}} 2^{\ell_\lambda} + \frac{256}{7} \cdot \left(4\mathrm{OPT} + \frac{165}{256}\sum_{\lambda \in \Lambda^{\mathrm{pf}}} 2^{\ell_\lambda}\right) \\
&= O(1) \cdot \mathrm{OPT} + O(1) \cdot \sum_{\lambda \in \Lambda^{\mathrm{pf}}} 2^{\ell_\lambda} + O(1) \cdot \sum_{\lambda \in \Lambda^{\mathrm{ps}}} 2^{\ell_\lambda}
\end{aligned}$$

□

Definition A.12 (primary interals and cylinders). For every service $\lambda \in \Lambda^{\mathrm{pf}} \cup \Lambda^{\mathrm{ps}}$, we define:

(1) The primary time interval $I_{\mathrm{p}}(\lambda) := (\min_{q \in Q^\star_\lambda} r_q, t_\lambda]$.
(2) The primary cylinder $\gamma_{\mathrm{p}}(\lambda) := \left(B\left(a_\lambda, 2^{\ell_\lambda - 2}\right), I_{\mathrm{p}}(\lambda)\right)$.

We also define $\Gamma^{\mathrm{pf}}, \Gamma^{\mathrm{ps}}$ to be the sets of all primary cylinders of services from $\Lambda^{\mathrm{pf}}, \Lambda^{\mathrm{ps}}$, respectively. We define $\Gamma^{\mathrm{p}} = \Gamma^{\mathrm{pf}} \cup \Gamma^{\mathrm{ps}}$. In addition, for every i we define Γ^{p}_i to be the subset of primary cylinders from Γ^{p} that belong to level-i services.

Definition A.13 ($D^*_{\mathrm{p},\lambda}$). For every primary service $\lambda \in \Lambda^{\mathrm{p}}$, let $R \subseteq Q^\star_\lambda$ be the set of requests unserved in the optimal solution at t_λ. Define $D^*_{\mathrm{p},\lambda} := \sum_{q \in R} y_q(t_\lambda)$. (Here, recall that t_λ refers to the time immediately before the service λ.)

Proposition A.14. *Let $\Lambda' \subseteq \Lambda^{\mathrm{pf}} \cup \Lambda^{\mathrm{ps}}$ be a set of services such that their primary cylinders are disjoint. Then it holds that $\sum_{\lambda \in \Lambda'} \left(c(\mathrm{OPT} \cap \gamma_{\mathrm{p}}(\lambda)) + D^*_{\mathrm{p},\lambda}\right) \leq \mathrm{OPT}$.*

Proof. Denote by $\mathrm{OPT}^m, \mathrm{OPT}^d$ the movement and delay costs of the optimal solution, respectively. One can observe, as in Observation 1, that $\sum_{\lambda \in \Lambda'} c(\mathrm{OPT} \cap \gamma_{\mathrm{c}}(\lambda)) \leq \mathrm{OPT}^m$. It remains to show that $\sum_{\lambda \in \Lambda'} D^*_{\mathrm{p},\lambda} \leq \mathrm{OPT}^d$.

Recall that the definition of $D^*_{\mathrm{p},\lambda}$ is $\sum_{q \in R} y_q(t_\lambda)$, where $R \subseteq Q^\star_\lambda$ is the subset of requests unserved by the optimal solution until t_λ. Note that the optimal solution indeed incurs $y_q(t_\lambda) = d_q(t_\lambda) - h_q t_\lambda$ delay for every request $q \in R$. Moreover, delay is never charged twice, as the service raises h_q to be $d_q(t_\lambda)$ at service λ. This completes the proof. □

Proposition A.15. *For every i, the set Γ^{p}_i is a set of disjoint cylinders.*

Proof. Assume for contradiction that there exist two services λ_1, λ_2 of level i such that $\gamma_{\mathrm{p}}(\lambda_1), \gamma_{\mathrm{p}}(\lambda_2)$ are not disjoint. As the cylinders' time intervals are not disjoint, without loss of generality, assume that $t_{\lambda_1} \in I_{\mathrm{p}}(\lambda_2)$. Since the cylinders are also not spatially disjoint, it holds that $\delta\left(a_{\lambda_1}, a_{\lambda_2}\right) \leq 2^{i-1}$. Now, from the definition of $I_{\mathrm{p}}(\lambda_2)$, there exists a request $q \in Q^\star_{\lambda_2}$ such that q is pending at t_{λ_1}. Now, note that since $q \in Q^\star_{\lambda_2}$, it holds that $\delta\left(q, a_{\lambda_2}\right) \leq 2^{i-3}$, which implies $\delta\left(q, a_{\lambda_1}\right) \leq 2^{i-1} + 2^{i-3} \leq 2^i$. Moreover, $\ell_q \leq i$ at t_{λ_1}. These facts imply that q was eligible for λ_1, but this would imply that q increases in level to $i+1$ after λ_1, in contradiction to $q \in Q^\star_{\lambda_2}$. □

Proposition A.16. *For every $\lambda \in \Lambda^{\mathrm{pf}}$, it holds that*

$$2^{\ell_\lambda - 8} \leq c\left(\mathrm{OPT} \cap \gamma_{\mathrm{p}}(\lambda)\right) + D^*_{\mathrm{p},\lambda}.$$

Proof. Consider the location a' to which the algorithm moved its server at the end of λ. From the definition of Λ^{pf}, we know that $a^*(t_\lambda) \notin B\left(a', 2^{\ell_\lambda - 7}\right)$. However, we also know from the definition of a' that at least $2^{\ell_\lambda - 4}$ of the residual delay of λ accumulated inside the ball $B\left(a', 2^{\ell_\lambda - 8}\right)$. Thus, at least one of the following holds:

(1) The optimal server visited $B\left(a', 2^{\ell_\lambda - 8}\right)$ during $I_{\mathrm{p}}(\lambda)$ and left by t_λ; thus, it incurred a moving cost of at least $2^{\ell_\lambda - 8}$ inside $B\left(a', 2^{\ell_\lambda - 7}\right)$. Now, note that $\delta(a', a) + 2^{\ell_\lambda - 7} \leq 2^{\ell_\lambda - 3} + 2^{\ell_\lambda - 8} + 2^{\ell_\lambda - 7} < 2^{\ell_\lambda - 2}$, and thus $B\left(a', 2^{\ell_\lambda - 7}\right) \subseteq B\left(a, 2^{\ell_\lambda - 2cx}\right)$. Thus, $c\left(\mathrm{OPT} \cap \gamma_{\mathrm{p}}(\lambda)\right) \geq 2^{\ell_\lambda - 8}$.

(2) The optimal server did not visit $B\big(a', 2^{\ell_\lambda-8}\big)$ during $I_{\mathrm{p}}(\lambda)$. In this case, it must be that $D^*_{\mathrm{p},\lambda} \geq 2^{\ell_\lambda-4}$.

In both cases, $2^{\ell_\lambda-8} \leq c\big(\mathrm{OPT} \cap \gamma_{\mathrm{p}}(\lambda)\big) + D^*_{\mathrm{p},\lambda}$. □

Proposition A.17. *For every $\lambda \in \Lambda^{\mathrm{ps}}$, it holds that*

$$2^{\ell_\lambda-8} \leq c\big(\mathrm{OPT} \cap \gamma_{\mathrm{p}}(\lambda)\big) + D^*_{\mathrm{p},\lambda}.$$

Proof. Let a^* be the location of the optimal server at t_λ. At least one of the following options holds:

(1) $a^* \notin B\big(a_\lambda, 1.5 \cdot 2^{\ell_\lambda-3}\big)$. In this case, we have the following subcases. Either the optimum did not visit $B\big(a_\lambda, 2^{\ell_\lambda-3}\big)$ during $I_{\mathrm{p}}(\lambda)$, in which case all requests in $Q^\star_\lambda$ remain unserved in OPT at t_λ, and $D^*_{\mathrm{p},\lambda} \geq 2^{\ell_\lambda-3}$; or, the optimal solution visited $B\big(a_\lambda, 2^{\ell_\lambda-3}\big)$ during $I_{\mathrm{p}}(\lambda)$, which in turn implies that $c\big(\mathrm{OPT} \cap \gamma_{\mathrm{p}}(\lambda)\big) \geq 2^{\ell_\lambda-4}$. In both cases, $c\big(\mathrm{OPT} \cap \gamma_{\mathrm{p}}(\lambda)\big) + D^*_{\mathrm{p},\lambda} \geq 2^{\ell_\lambda-4}$.

(2) $a^* \in B\big(a_\lambda, 1.5 \cdot 2^{\ell_\lambda-3}\big)$. In this case, consider $B\big(a^*, 2^{\ell_\lambda-8}\big)$: since $\lambda \in \Lambda^{\mathrm{ps}}$, it must be that the residual delay of λ inside $B\big(a^*, 2^{\ell_\lambda-8}\big)$ is at most $2^{\ell_\lambda-4}$. But, the total residual delay of λ is at least $2^{\ell_\lambda-3}$. Thus, at least $2^{\ell_\lambda-4}$ residual delay of λ is outside $B\big(a^*, 2^{\ell_\lambda-8}\big)$. If OPT was outside $B\big(a^*, 2^{\ell_\lambda-8}\big)$ during $I_{\mathrm{p}}(\lambda)$, it must be that $c\big(\mathrm{OPT} \cap \gamma_{\mathrm{p}}(\lambda)\big) \geq 2^{\ell_\lambda-8}$ (for this, note that $B\big(a^*, 2^{\ell_\lambda-8}\big) \subseteq B\big(a_\lambda, 2^{\ell_\lambda-2}\big)$ and thus inside $\gamma_{\mathrm{p}}(\lambda)$). Otherwise, OPT remained in $B\big(a^*, 2^{\ell_\lambda-8}\big)$ during $I_{\mathrm{p}}(\lambda)$, and thus $D^*_{\mathrm{p},\lambda} \geq 2^{\ell_\lambda-4}$. In both cases, $c\big(\mathrm{OPT} \cap \gamma_{\mathrm{p}}(\lambda)\big) + D^*_{\mathrm{p},\lambda} \geq 2^{\ell_\lambda-8}$.

In all cases, $c\big(\mathrm{OPT} \cap \gamma_{\mathrm{p}}(\lambda)\big) + D^*_{\mathrm{p},\lambda} \geq 2^{\ell_\lambda-8}$. □

Lemma A.18. $\sum_{\lambda \in \Lambda^{\mathrm{pf}} \cup \Lambda^{\mathrm{ps}}} 2^{\ell_\lambda} \leq O(\log n) \cdot \mathrm{OPT}$.

Proof. Define $\rho = 2^9 \cdot n^2$. Combining Propositions A.16 and A.17 and Corollary B.3, for every service $\lambda \in \Lambda^{\mathrm{pf}} \cup \Lambda^{\mathrm{ps}}$, we have that

$$\begin{aligned} 2^{\ell_\lambda-8} &\leq c(\mathrm{OPT} \cap \gamma_\lambda) + D^*_{\mathrm{p},\lambda} \\ &\leq c\Big(\gamma^\rho_\lambda \cap \mathrm{OPT}\Big) + 2 \cdot 2^{\ell_\lambda-1} \cdot n^2/\rho + D^*_{\mathrm{p},\lambda} \\ &\leq c\Big(\gamma^\rho_\lambda \cap \mathrm{OPT}\Big) + 2^{\ell_\lambda-9} + D^*_{\mathrm{p},\lambda} \end{aligned}$$

Simplifying, we get $2^{\ell_\lambda-9} \leq c\Big(\gamma^\rho_\lambda \cap \mathrm{OPT}\Big) + D^*_{\mathrm{p},\lambda}$. Summing over all $\lambda \in \Lambda^{\mathrm{pf}} \cup \Lambda^{\mathrm{ps}}$, we have

$$\sum_{\lambda \in \Lambda^{\mathrm{pf}} \cup \Lambda^{\mathrm{ps}}} 2^{\ell_\lambda} \leq O(1) \cdot \sum_{\lambda \in \Lambda^{\mathrm{pf}} \cup \Lambda^{\mathrm{ps}}} \Big(c\Big(\gamma^\rho_\lambda \cap \mathrm{OPT}\Big) + D^*_{\mathrm{p},\lambda} \Big).$$

Note that for every i, the cylinders of Γ^{p}_i are disjoint (Proposition A.15). Thus, defining $\mathscr{H} := \Big\{ \gamma^\rho_\lambda | \gamma_\lambda \in \Gamma^{\mathrm{c}} \Big\}$, Proposition B.4 implies that $\mathscr{H}$ can be partitioned into $O(\log n)$ disjoint sets. Proposition A.24 thus implies that $\sum_{\lambda \in \Lambda^{\mathrm{c}}} 2^{\ell_\lambda} \leq O(\log n) \cdot \mathrm{OPT}$, completing the proof. □

Bounding Certified Services. In this subsection, we would like to prove the following.

Lemma A.19. $\sum_{\lambda \in \Lambda^{\mathrm{c}}} 2^{\ell_\lambda} \leq O(\log n) \cdot \mathrm{OPT}$.

Definition A.20. Let $\lambda \in \Lambda^{\mathrm{c}}$ be a certified service. Let $\lambda' \in \Lambda^{\mathrm{c}}$ be the certified service with maximum $\tau_{\lambda'}$ subject to $\ell_{\lambda'} = \ell_\lambda$, $\tau_{\lambda'} \leq t_\lambda$ and $\delta(q_{\lambda'}, q_\lambda) < 6 \cdot 2^{\ell_\lambda}$. We define:

(1) The time $\sigma_\lambda := \tau_{\lambda'}$ if λ' exists (otherwise, define $\sigma_\lambda = -\infty$).

(2) The time interval $I_{\mathrm{c}}(\lambda) := (\sigma_\lambda, \tau_\lambda]$; note that $t_\lambda \in I_{\mathrm{c}}(\lambda)$.

Definition A.21 ($\pi_{\mathrm{c},\lambda}$ and $D^*_{\mathrm{c},\lambda}$). For every certified service $\lambda \in \Lambda^{\mathrm{c}}$:

(1) Define $\pi_{\mathrm{c},\lambda}$ on requests $q \in E_\lambda$ such that

$$\pi_{\mathrm{c},\lambda}(q) := \big(d_q(\tau_\lambda) - \max\big\{d_q(t_\lambda), h_q(t_\lambda)\big\}\big)^+.$$

(Here, recall that t_λ refers to the time immediately before the service λ.)

(2) Let $R \subseteq E_\lambda$ be the subset of λ-eligible requests unserved in the optimal solution at τ_λ. Define $D^*_{\mathrm{c},\lambda} := \sum_{q \in R} \pi_{\mathrm{c},\lambda}(q)$.

Proposition A.22 (analogue of Proposition 3.19). *Let services $\lambda_1, \lambda_2 \in \Lambda^{\mathrm{c}}$ be such that $\ell_{\lambda_1} = \ell_{\lambda_2} = \ell$ and $\delta\big(a_{\lambda_1}, a_{\lambda_2}\big) \leq 6 \cdot 2^\ell$. Assuming WLOG that $t_{\lambda_1} < t_{\lambda_2}$, and letting λ be the service that made λ_1 certified, it holds that $t_\lambda \in (\tau_{\lambda_1}, t_{\lambda_2}]$. (In particular, $\tau_{\lambda_1} < t_{\lambda_2}$.)*

Proof. The two possible cases which contradict our proposition are that $t_\lambda \leq \tau_{\lambda_1}$ or that $t_\lambda > t_{\lambda_2}$. First, we prove that $t_\lambda > \tau_{\lambda_1}$ Consider the triggering request $q \in Q^\star_\lambda$ that made λ certify λ_1: this request was a witness for λ_1, and thus in E_{λ_1}; however, λ_1 maintains that eligible requests will not accumulate residual delay until after time τ_{λ_1}. But, requests in $Q^\star_\lambda$ have positive residual delay at t_λ; thus $t_\lambda > \tau_{\lambda_1}$.

Now, assume for contradiction that $t_\lambda > t_{\lambda_2}$. Consider the service λ' which certified λ_2; it must also be the case that $t_{\lambda'} > t_{\lambda_2}$. Suppose that $t_\lambda < t_{\lambda'}$. In this case, observe that all witnesses for λ_2 at t_λ are in $B\big(a_{\lambda_2}, 2^\ell\big)$; but it holds through triangle inequality that

$$\delta\big(a_{\lambda_2}, a_\lambda\big) \leq \delta\big(a_{\lambda_2}, a_{\lambda_1}\big) + \delta\big(a_{\lambda_1}, q\big) + \delta\big(q, a_\lambda\big) \leq 6 \cdot 2^\ell + 2^\ell + 2^{\ell_\lambda-3}$$

Using Proposition A.5, $\ell_\lambda \leq \ell + 5$, yielding that $\delta\big(a_{\lambda_2}, a_\lambda\big) \leq 11 \cdot 2^\ell$. This implies that $B\big(a_{\lambda_2}, 2^\ell\big) \subseteq B\big(a_\lambda, 12 \cdot 2^\ell\big)$. However, Proposition A.5 again implies $\ell_\lambda \geq \ell + 4$, and thus $B\big(a_{\lambda_2}, 2^\ell\big) \subseteq B\big(a_\lambda, 2^{\ell_\lambda}\big)$. Combine this with the fact that the level of all witnesses for λ_2 at t_λ is at most $\ell+1$ which is less than ℓ_λ; we thus obtain that all witnesses to λ_2 at t_λ are in E_λ. But, after λ, these witnesses would no longer be witnesses for λ_2, in contradiction to one of them triggering λ' and certifying λ_2. Similarly, if $t_\lambda > t_{\lambda'}$, the service λ' would leave no witnesses for λ_1 to trigger λ, which is a contradiction. □

Definition A.23 (certified cylinders). For a certified service $\lambda \in \Lambda^{\mathrm{c}}$, define the certified cylinder $\gamma_{\mathrm{c}}(\lambda) := \big(B\big(q_\lambda, 3 \cdot 2^{\ell_q}\big), I_{\mathrm{c}}(\lambda)\big)$. Define Γ^{c} to be the set of all certified cylinders; in addition, for every i define Γ^{c}_i to be the set of certified cylinders formed from level-i services.

Proposition A.24. *Let $\Lambda' \subseteq \Lambda^{\mathrm{c}}$ be a set of services such that their certified cylinders are disjoint. Then it holds that*

$$\sum_{\lambda \in \Lambda'} \Big(c(\mathrm{OPT} \cap \gamma_{\mathrm{c}}(\lambda)) + D^*_{\mathrm{c},\lambda} \Big) \leq \mathrm{OPT}.$$

Proof. Denote by $\mathrm{OPT}^m, \mathrm{OPT}^d$ the movement and delay costs of the optimal solution, respectively. One can observe, as in Observation 1, that $\sum_{\lambda \in \Lambda'} c(\mathrm{OPT} \cap \gamma_{\mathrm{c}}(\lambda)) \leq \mathrm{OPT}^m$. It remains to show that $\sum_{\lambda \in \Lambda'} D^*_{\mathrm{c},\lambda} \leq \mathrm{OPT}^d$.

Recall that the definition of $D^*_{\mathrm{c},\lambda}$ is $\sum_{q \in R} \pi_{\mathrm{c},\lambda}(q)$, where $R \subseteq E_\lambda$ is the set of requests unserved by the optimal solution until τ_λ. For every $q \in R$, note that $h'_q := \max\big\{d_q(t_\lambda), h_q(t_\lambda)\big\}$ is exactly the value of the counter h_q after Line 13 of λ; thus, $\pi_{c,\lambda}(q) = \left(d_q(t_\lambda) - h'_q\right)^+$. Let $t' \in [t_\lambda, \tau_\lambda]$ be the point in time in which $d_q(t') = h'_q$; the optimal solution incurs a delay cost of $d_q(\tau_\lambda) - h'_q$ during the interval $[t', \tau_\lambda]$, and thus the charging $D^*_{c,\lambda}$ is valid. Moreover, λ either serves q, or raises h_q to at least $d_q(\tau_\lambda)$ (in Line 20); thus, the delay of q during $[t', \tau_\lambda]$ is only charged once to the optimal solution. This completes the proof. □

Proposition A.25. *For every i, Γ^c_i is a set of disjoint cylinders.*

Proof. The proposition results from Proposition A.22 in the same way that Proposition 3.21 results from Proposition 3.19. □

Proposition A.26. *For every certified service $\lambda \in \Lambda^c$ and request $q \in E_\lambda$, it holds that $r_q > \sigma_\lambda$.*

Proof. If $\sigma_\lambda = -\infty$ then we are done. Otherwise, there exists a certified service $\lambda' \in \Lambda^c$ such that $\ell_{\lambda'} = \ell_\lambda$, $\tau_{\lambda'} = \sigma_\lambda$, and $\delta\left(a_\lambda, a'_\lambda\right) \le 6 \cdot 2^{\ell_\lambda}$. Define $\ell := \ell_\lambda$. Using Proposition A.22, the service λ'' that certified λ' occured in the interval $(\tau_{\lambda'}, t_\lambda]$. Thus, there must exist a witness request that made λ'' certify λ'; thus, there exists a request in $E_{\lambda'} \cap Q^\star_{\lambda''}$. But this means that $\delta(a_{\lambda'}, a_{\lambda''}) \le 2^{\ell_{\lambda'}} + 2^{\ell_{\lambda''}-3} \le 5 \cdot 2^\ell$, where the second inequality uses Proposition A.5 for $\ell_{\lambda''} \le \ell_\lambda + 5$.

Assume for contradiction that $r_q \le \sigma_\lambda = \tau_{\lambda'}$. Thus, q is pending at $t_{\lambda''}$, and also has level at most ℓ (since $q \in E_\lambda$). In addition we have:

$$\begin{aligned}\delta(q, a_{\lambda''}) &\le \delta(q, a_\lambda) + \delta(a_\lambda, a_{\lambda'}) + \delta(a_{\lambda'}, a_{\lambda''})\\ &\le 2^\ell + 6 \cdot 2^\ell + 5 \cdot 2^\ell = 12 \cdot 2^\ell \le 2^{\ell_{\lambda''}}\end{aligned}$$

where the final inequality uses Proposition A.5 to claim that $\ell_{\lambda''} \ge \ell_\lambda + 4$. Thus, $q \in E_{\lambda''}$, and since it is not served by λ'', its level after λ'' is at least $\ell_{\lambda''} + 1 \ge \ell + 5$; but this contradicts the level of q being at most ℓ at t_λ. This completes the proof of the proposition. □

Proposition A.27. *For every certified service $\lambda \in \Lambda^c$, it holds that $2^{\ell_\lambda} \le 2c(\mathrm{OPT} \cap \gamma_c(\lambda)) + D^*_{c,\lambda}$.*

Proof. Denote by $R \subseteq E_\lambda$ the set of requests in E_λ whose location was visited by during $I_c(\lambda)$. Thus, using Proposition 3.22, it holds that $\mathrm{ST}^*(R) \le 2c\left(\mathrm{OPT} \cap 3 \cdot 2^{\ell_\lambda}\right)$. Now, note that $\mathrm{ST}^*(R \cup \{a_\lambda\}) \le \mathrm{ST}^*(R) + 2^{\ell_\lambda}$. Finally, note that the prize-collecting Steiner tree problem whose solution is traversed by the algorithm uses the penalty function $\pi_{c,\lambda}$. Combining, we get

$$\begin{aligned}2^{\ell_\lambda+1} &\le \mathrm{PCST}^*(E_\lambda, \pi_\lambda; a_\lambda)\\ &\le \mathrm{ST}^*(R \cup \{a_\lambda\}) + \sum_{q \in E_\lambda \backslash R} \pi_{c,\lambda}(q)\\ &\le \mathrm{ST}^*(R) + 2^{\ell_\lambda} + D^*_{c,\lambda}\\ &\le 2c\left(\mathrm{OPT} \cap 3 \cdot 2^{\ell_\lambda}\right) + 2^{\ell_\lambda} + D^*_{c,\lambda}\end{aligned}$$

Where the first inequality stems from Line 16 together with the fact that PCST is a 3-approximation and the second inequality is since serving only R is a feasible solution to $\mathrm{PCST}^*(E_\lambda, \pi_\lambda; a_\lambda)$. Simplifying, we get that $2^{\ell_\lambda} \le 2c(\mathrm{OPT} \cap \gamma_c(\lambda)) + D^*_{c,\lambda}$. □

Proof of Lemma A.19. Define $\rho = 24 \cdot n^2$. Combining Proposition A.27 and Corollary B.3, for every cylinder $\gamma_\lambda \in \Gamma^c$, we have

$$\begin{aligned}2^{\ell_\lambda} &\le 2c(\mathrm{OPT} \cap \gamma_\lambda) + D^*_{c,\lambda}\\ &\le 2c\left(\gamma^\rho_\lambda \cap \mathrm{OPT}\right) + 4 \cdot 3 \cdot 2^{\ell_\lambda} \cdot n^2/\rho + D^*_{c,\lambda}\\ &\le 2c\left(\gamma^\rho_\lambda \cap \mathrm{OPT}\right) + 2^{\ell_\lambda - 1} + D^*_{c,\lambda}\end{aligned}$$

Thus, $\sum_{\lambda \in \Lambda^c} 2^{\ell_\lambda} \le O(1) \cdot \sum_{\lambda \in \Lambda}(c(\mathrm{OPT} \cap \gamma_\lambda) + D^*_{c,\lambda})$. Note that for every i, the cylinders of Γ^c_i are disjoint (Proposition A.25). Thus, defining $\mathscr{H} := \left\{\gamma^\rho_\lambda | \gamma_\lambda \in \Gamma^c\right\}$, Proposition B.4 implies that $\mathscr{H}$ can be partitioned into $O(\log n)$ disjoint sets. Proposition A.24 thus implies that $\sum_{\lambda \in \Lambda^c} 2^{\ell_\lambda} \le O(\log n) \cdot \mathrm{OPT}$, completing the proof. □

Proof of Theorem A.1. The following holds:

$$\begin{aligned}\mathrm{ALG} &\le O(1) \cdot \left(\sum_{\lambda \in \Lambda^c} 2^{\ell_\lambda} + \sum_{\lambda \in \Lambda^{pf}} 2^{\ell_\lambda} + \sum_{\lambda \in \Lambda^{ps}} 2^{\ell_\lambda} + \mathrm{OPT}\right)\\ &\le O(\log n) \cdot \mathrm{OPT}\end{aligned}$$

where the first inequality uses Propositions A.10 and A.11, and the second inequality uses Lemmas A.18 and A.19 □

B IMPROVED ANALYSIS THROUGH PERFORATED CYLINDERS

Perforated balls and cylinders.

Definition B.1 (perforated balls and cylinders). For every point $v \in G$, radius r, and number $\rho > 1$, define the perforated ball

$$B^\rho(v, r) := B(v, r) - \bigcup_{v' \in G} B\left(v', \frac{r}{\rho}\right).$$

In addition, given a cylinder $\gamma = (B(v, r), I)$, define the perforated cylinder $\gamma^\rho := (B^\rho(v, r), I)$.

Proposition B.2. *For every subgraph G', and every choice of v, r, ρ, it holds that $c(G' \cap B(v, r)) \le c(G' \cap B^\rho(v, r)) + \frac{2rn^2}{\rho}$.*

Proof. Consider every edge e in G'. The total weight of the edge e contained in balls of radius r/ρ is at most $2r/\rho$ (specifically, this weight is contained in the intersection with the balls centered in the two endpoints of the edge). The fact that the number of edges in G' is at most n^2 completes the proof. □

The following corollary follows immediately.

Corollary B.3. *For every cylinder $\gamma = (B(v, r), I)$ and parameter ρ, it holds that*

$$c(\mathrm{OPT} \cap \gamma) \le c\left(\mathrm{OPT} \cap \gamma^\rho\right) + \frac{2rn^2}{\rho}$$

Proposition B.4. *Suppose that for every integer i, Γ_i is some set of disjoint cylinders of the form $(B(v, x), I)$ where $\lceil \log x \rceil = i$, and define $\Gamma = \bigcup_i \Gamma_i$. Then for every parameter $\rho \ge 2$, and defining $\mathscr{H} := \{\gamma^\rho | \gamma \in \Gamma\}$, the set $\mathscr{H}$ can be partitioned into $O(\log \rho)$ sets of disjoint cylinders.*

Proof. Define $b = \lceil \log \rho \rceil + 1$, and define $\mathscr{H}_i = \{\gamma^\rho | \gamma \in \Gamma_i\}$. For every $i \in [b]$, define $\overline{\mathscr{H}}_i = \bigcup_{j \in i + b\mathbb{Z}} \mathscr{H}_j$; note that $\mathscr{H}$ is partitioned into those b sets.

It remains to show that $\overline{\mathscr{H}}_i$ is a set of disjoint cylinders for every i. Consider two cylinders $\gamma_1^\rho, \gamma_2^\rho \in \overline{\mathscr{H}}_i$, denote the centers of their balls by v_1, v_2, and denote the radii of their balls by r_1, r_2, respectively. If $\lceil \log r_1 \rceil = \lceil \log r_2 \rceil = i$, then $\gamma_1^\rho, \gamma_2^\rho \in \mathscr{H}_i$; in this case, they must be disjoint: Γ_i is a set of disjoint cylinders, and $\gamma^\rho \subseteq \gamma$ for every γ. Otherwise, $r_1 \neq r_2$; assume WLOG that $\lceil \log r_1 \rceil \geq \lceil \log r_2 \rceil + b$. Now, consider that $B^\rho(v_1, r_1)$ and $B(v_2, r_1/\rho)$ are disjoint, by construction; now note that $r_2 \leq r_1/2^{b-1} \leq r_1/\rho$, and thus $B^\rho(v_2, r_2) \subseteq B(v_2, r_1/\rho)$, which shows that $\gamma_1^\rho, \gamma_2^\rho$ are disjoint. Overall, we proved that $\overline{\mathscr{H}}_i$ is a disjoint set. □

Proof of Lemma 3.25. Define $\rho = 2^6 n^2$. Combining Proposition 3.15 and Corollary B.3, for every cylinder $\gamma_\lambda \in \Gamma^{\mathrm{p}}$, we have

$$\begin{aligned} 2^{\ell_\lambda - 6} &\leq c(\mathrm{OPT} \cap \gamma_\lambda) \\ &\leq c\left(\gamma_\lambda^\rho \cap \mathrm{OPT}\right) + 2 \cdot 2^{\ell_\lambda - 2} \cdot n^2/\rho \\ &\leq c\left(\gamma_\lambda^\rho \cap \mathrm{OPT}\right) + 2^{\ell_\lambda - 7}. \end{aligned}$$

Simplifying, we get $2^{\ell_\lambda - 7} \leq c\left(\gamma_\lambda^\rho \cap \mathrm{OPT}\right)$. Defining the set of cylinders $\mathscr{H}^{\mathrm{pf}} := \left\{\gamma_\lambda^\rho \middle| \gamma_\lambda \in \Gamma^{\mathrm{p}}\right\}$, we have $\sum_{\lambda \in \Lambda^{\mathrm{pf}}} 2^{\ell_\lambda} \leq O(1) \cdot \sum_{\lambda \in \Lambda^{\mathrm{pf}}} c\left(\mathrm{OPT} \cap \gamma_\lambda^\rho\right)$. Now, using Proposition B.4, we know that $\mathscr{H}^{\mathrm{pf}}$ can be partitioned into $O(\log \rho) = O(\log n)$ sets of disjoint cylinders. Thus, $\sum_{\lambda \in \Lambda^{\mathrm{pf}}} c\left(\mathrm{OPT} \cap \gamma_\lambda^\rho\right) \leq O(\log n) \cdot \mathrm{OPT}$. □

Proof of Lemma 3.26. Define $\rho = 24 \cdot n^2$. Combining Proposition 3.24 and Corollary B.3, for every cylinder $\gamma_\lambda \in \Gamma^{\mathrm{c}}$, we have

$$\begin{aligned} 2^{\ell_\lambda - 1} &\leq c(\mathrm{OPT} \cap \gamma_\lambda) \\ &\leq c\left(\gamma_\lambda^\rho \cap \mathrm{OPT}\right) + 2 \cdot 3 \cdot 2^{\ell_\lambda} \cdot n^2/\rho \\ &\leq c\left(\gamma_\lambda^\rho \cap \mathrm{OPT}\right) + 2^{\ell_\lambda - 2}. \end{aligned}$$

Simplifying, $2^{\ell_\lambda - 2} \leq c\left(\gamma_\lambda^\rho \cap \mathrm{OPT}\right)$. Defining $\mathscr{H} := \left\{\gamma_\lambda^\rho \middle| \gamma_\lambda \in \Gamma^{\mathrm{c}}\right\}$, we have $\sum_{\lambda \in \Lambda^{\mathrm{c}}} 2^{\ell_\lambda} \leq O(1) \cdot \sum_{\lambda \in \Lambda^{\mathrm{c}}} c\left(\mathrm{OPT} \cap \gamma_\lambda^\rho\right)$. Proposition B.4 yields that $\mathscr{H}$ can be partitioned into $O(\log \rho) = O(\log n)$ sets of disjoint cylinders. Thus, $\sum_{\lambda \in \Lambda^{\mathrm{c}}} c\left(\mathrm{OPT} \cap \gamma_\lambda^\rho\right) \leq O(\log n) \cdot \mathrm{OPT}$. □

REFERENCES

[1] Yossi Azar, Arun Ganesh, Rong Ge, and Debmalya Panigrahi. 2017. Online service with delay. In *Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2017, Montreal, QC, Canada, June 19-23, 2017*. 551–563. https://doi.org/10.1145/3055399.3055475

[2] Yossi Azar and Noam Touitou. 2019. General Framework for Metric Optimization Problems with Delay or with Deadlines. In *60th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2019, Baltimore, Maryland, USA, November 9-12, 2019*. 60–71. https://doi.org/10.1109/FOCS.2019.00013

[3] Yossi Azar and Noam Touitou. 2020. Beyond Tree Embeddings - a Deterministic Framework for Network Design with Deadlines or Delay. In *61st IEEE Annual Symposium on Foundations of Computer Science, FOCS 2020, Durham, NC, USA, November 16-19, 2020*. IEEE, 1368–1379. https://doi.org/10.1109/FOCS46700.2020.00129

[4] Nikhil Bansal, Niv Buchbinder, Aleksander Madry, and Joseph Naor. 2011. A Polylogarithmic-Competitive Algorithm for the k-Server Problem. In *IEEE 52nd Annual Symposium on Foundations of Computer Science, FOCS 2011, Palm Springs, CA, USA, October 22-25, 2011*, Rafail Ostrovsky (Ed.). IEEE Computer Society, 267–276. https://doi.org/10.1109/FOCS.2011.63

[5] Marcin Bienkowski, Martin Böhm, Jaroslaw Byrka, Marek Chrobak, Christoph Dürr, Lukáš Folwarczný, Lukasz Jez, Jiri Sgall, Nguyen Kim Thang, and Pavel Veselý. 2016. Online Algorithms for Multi-Level Aggregation. In *24th Annual European Symposium on Algorithms, ESA 2016, August 22-24, 2016, Aarhus, Denmark*. 12:1–12:17. https://doi.org/10.4230/LIPIcs.ESA.2016.12

[6] Marcin Bienkowski, Jaroslaw Byrka, Marek Chrobak, Lukasz Jez, Dorian Nogneng, and Jirí Sgall. 2014. Better Approximation Bounds for the Joint Replenishment Problem. In *Proceedings of the Twenty-Fifth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2014, Portland, Oregon, USA, January 5-7, 2014*. 42–54. https://doi.org/10.1137/1.9781611973402.4

[7] Marcin Bienkowski, Artur Kraska, and Pawel Schmidt. 2018. Online Service with Delay on a Line. In *Structural Information and Communication Complexity - 25th International Colloquium, SIROCCO 2018, Ma'ale HaHamisha, Israel, June 18-21, 2018, Revised Selected Papers*. 237–248. https://doi.org/10.1007/978-3-030-01325-7_22

[8] Carlos Fisch Brito, Elias Koutsoupias, and Shailesh Vaya. 2012. Competitive Analysis of Organization Networks or Multicast Acknowledgement: How Much to Wait? *Algorithmica* 64, 4 (2012), 584–605. https://doi.org/10.1007/s00453-011-9567-5

[9] Sébastien Bubeck, Michael B. Cohen, Yin Tat Lee, James R. Lee, and Aleksander Madry. 2018. k-server via multiscale entropic regularization. In *Proceedings of the 50th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2018, Los Angeles, CA, USA, June 25-29, 2018*, Ilias Diakonikolas, David Kempe, and Monika Henzinger (Eds.). ACM, 3–16. https://doi.org/10.1145/3188745.3188798

[10] Niv Buchbinder, Moran Feldman, Joseph (Seffi) Naor, and Ohad Talmon. 2017. O(depth)-Competitive Algorithm for Online Multi-level Aggregation. In *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2017, Barcelona, Spain, Hotel Porta Fira, January 16-19*. 1235–1244. https://doi.org/10.1137/1.9781611974782.80

[11] Niv Buchbinder, Kamal Jain, and Joseph Naor. 2007. Online Primal-Dual Algorithms for Maximizing Ad-Auctions Revenue. In *Algorithms - ESA 2007, 15th Annual European Symposium, Eilat, Israel, October 8-10, 2007, Proceedings*. 253–264. https://doi.org/10.1007/978-3-540-75520-3_24

[12] Niv Buchbinder, Tracy Kimbrel, Retsef Levi, Konstantin Makarychev, and Maxim Sviridenko. 2008. Online make-to-order joint replenishment model: primal dual competitive algorithms. In *Proceedings of the Nineteenth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2008, San Francisco, California, USA, January 20-22, 2008*. 952–961. http://dl.acm.org/citation.cfm?id=1347082.1347186

[13] Ryder Chen, Jahanvi Khatkar, and Seeun William Umboh. 2022. Online Weighted Cardinality Joint Replenishment Problem with Delay. In *49th International Colloquium on Automata, Languages, and Programming, ICALP 2022, July 4-8, 2022, Paris, France (LIPIcs, Vol. 229)*, Mikolaj Bojanczyk, Emanuela Merelli, and David P. Woodruff (Eds.). Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 40:1–40:18. https://doi.org/10.4230/LIPIcs.ICALP.2022.40

[14] Daniel R. Dooly, Sally A. Goldman, and Stephen D. Scott. 1998. TCP Dynamic Acknowledgment Delay: Theory and Practice (Extended Abstract). In *Proceedings of the Thirtieth Annual ACM Symposium on the Theory of Computing, Dallas, Texas, USA, May 23-26, 1998*. 389–398. https://doi.org/10.1145/276698.276792

[15] Anupam Gupta, Amit Kumar, and Debmalya Panigrahi. 2021. A Hitting Set Relaxation for k-Server and an Extension to Time-Windows. In *62nd IEEE Annual Symposium on Foundations of Computer Science, FOCS 2021, Denver, CO, USA, February 7-10, 2022*. IEEE, 504–515. https://doi.org/10.1109/FOCS52979.2021.00057

[16] Anupam Gupta, Amit Kumar, and Debmalya Panigrahi. 2022. Caching with Time Windows and Delays. *SIAM J. Comput.* 51, 4 (2022), 975–1017. https://doi.org/10.1137/20m1346286

[17] Mohammad Taghi Hajiaghayi and Kamal Jain. 2006. The Prize-collecting Generalized Steiner Tree Problem via a New Approach of Primal-dual Schema. In *Proceedings of the Seventeenth Annual ACM-SIAM Symposium on Discrete Algorithm* (Miami, Florida) *(SODA '06)*. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 631–640. http://dl.acm.org/citation.cfm?id=1109557.1109626

[18] Anna R. Karlin, Claire Kenyon, and Dana Randall. 2003. Dynamic TCP Acknowledgement and Other Stories about e/(e-1). *Algorithmica* 36, 3 (2003), 209–224.

[19] Lawrence T. Kou, George Markowsky, and Leonard Berman. 1981. A Fast Algorithm for Steiner Trees. *Acta Informatica* 15 (1981), 141–145. https://doi.org/10.1007/BF00288961

[20] Elias Koutsoupias and Christos H. Papadimitriou. 1995. On the k-Server Conjecture. *J. ACM* 42, 5 (1995), 971–983. https://doi.org/10.1145/210118.210128

[21] Mark S. Manasse, Lyle A. McGeoch, and Daniel Dominic Sleator. 1990. Competitive Algorithms for Server Problems. *J. Algorithms* 11, 2 (1990), 208–230. https://doi.org/10.1016/0196-6774(90)90003-W

Received 2022-11-07; accepted 2023-02-06