Nearly-Optimal Algorithm for Non-Clairvoyant Service with Delay

Noam Touitou
Unaffiliated, Tel Aviv, Israel

Abstract

We consider the online service with delay problem, in which a server traverses a metric space to serve requests that arrive over time. Requests gather individual delay cost while awaiting service, penalizing service latency; an algorithm seeks to minimize both its movement cost and the total delay cost. Algorithms for the problem (on general metric spaces) are only known for the clairvoyant model, where the algorithm knows future delay cost in advance (e.g., Azar et al., STOC'17; Azar and Touitou, FOCS'19; Touitou, STOC'23). However, in the non-clairvoyant setting, only negative results are known: where n is the size of the metric space and m is the number of requests, there are lower bounds of $\Omega(\sqrt{n})$ and $\Omega(\sqrt{m})$ on competitiveness (Azar et al., STOC'17), that hold even for randomized algorithms (Le et al., SODA'23).

In this paper, we present the first algorithm for non-clairvoyant online service with delay. Our algorithm is deterministic and $O\left(\min\left\{\sqrt{n}\log n, \sqrt{m}\log m\right\}\right)$-competitive; combined with the lower bounds of Azar et al., this settles the correct competitive ratio for the problem up to logarithmic factors, in terms of both n and m.

2012 ACM Subject Classification Theory of computation → K-server algorithms; Theory of computation → Online algorithms

Keywords and phrases Online, Delay, Deadlines, k-server, Non-clairvoyant

Digital Object Identifier 10.4230/LIPIcs.STACS.2025.74

1 Introduction

In online service with delay, or OSD, a server exists on a metric space of n points. Requests arrive over time, where every request is associated with a point in the metric space which the server must visit to satisfy the request. The algorithm may move the server at any moment in time, at a cost which is the distance traveled by the server in the metric space (the movement is instantaneous). In addition, requests accumulate delay cost while pending, motivating the algorithm to serve requests promptly. The goal of the algorithm is to minimize the sum of total movement cost and total delay cost over the given input.

In this model, a crucial choice is whether the online algorithm becomes aware of a request's future delay cost upon its release (*clairvoyant* model) or only knows delay cost accumulated in the past (*non-clairvoyant* model). Azar et al. [4], who introduced the problem of online service with delay, presented an algorithm for the clairvoyant model of polylogarithmic competitiveness in the size of the metric space n; specifically, $O(\log^4 n)$-competitiveness. This was later improved to $O(\log^2 n)$-competitiveness [6], and then to $O(\log(\min\{n, m\}))$-competitiveness [35], where m is the number of requests in the online input.

In the non-clairvoyant model, however, there are no known positive results. Azar et al. [4] presented $\Omega(\sqrt{n})$ and $\Omega(\sqrt{m})$ lower bounds on the competitiveness of any deterministic algorithm; this bound also holds for randomized algorithms [31]. (Note that the lower bound in [31] is stated for the joint replenishment problem, a special case of online service with delay.)

1.1 Our Results

In this paper, we present the first non-clairvoyant algorithm for online service with delay. Define k to be the number of locations on which requests are released in the input; we prove the following theorem.

▶ **Theorem 1.** *There exists a polynomial-time, deterministic, non-clairvoyant algorithm for online service with delay which is $O(\sqrt{k}\log k)$-competitive.*

In particular, note that $k \leq \min(n, m)$; thus, Theorem 1 also yields an $O\big(\min\{\sqrt{n}\log n, \sqrt{m}\log m\}\big)$ competitiveness bound. Recalling the $\Omega(\sqrt{n}), \Omega(\sqrt{m})$ lower bounds on competitiveness of [4] for non-clairvoyant algorithms for OSD, we conclude that our competitive ratio is tight up to a logarithmic factor.

Another problem of interest is online service with *deadlines*. In this problem, instead of accumulating delay cost, every request has an associated deadline by which it must be served. Online service with deadlines is a special case of online service with delay; intuitively, a request with a deadline corresponds to a request with delay that goes from zero to infinity at the time of the deadline. Yet, online service with deadlines has been studied explicitly in the past (e.g., [24, 25, 35]). Theorem 1 for online service with delay also yields Theorem 2 for online service with deadlines as a corollary, providing the first non-clairvoyant algorithm for online service with deadlines.

▶ **Theorem 2.** *There exists a polynomial-time, deterministic, non-clairvoyant algorithm for online service with deadlines which is $O(\sqrt{k}\log k)$-competitive.*

1.2 Related Work

A class of problems related to online service with delay is online network design with delay, where connectivity requests are handled by transmitting items. Such problems include TCP Acknowledgement [21, 29, 17, 28], joint replenishment [18, 15, 11, 19], multilevel aggregation [9, 16, 31, 6, 8], facility location [6, 7, 10], and set cover with delay [2, 33, 31]. In [7], a general framework for such problems in the clairvoyant model was introduced, yielding logarithmic competitiveness.

While some problems, such as set cover with delay, retain polylogarithmic competitiveness in the non-clairvoyant setting [2], others become much harder. This is the case for joint replenishment, facility location and multilevel aggregation with delay, which have $\Omega(\sqrt{n})$ and $\Omega(\sqrt{m})$ lower bounds on the competitiveness of any randomized algorithm [4, 31]. As online service with delay is a generalization of the joint replenishment and multilevel aggregation problems, these lower bounds extend also to non-clairvoyant service with delay. An $O(\min(\sqrt{n\log n}, \sqrt{m\log m}))$-competitive framework for non-clairvoyant network design was introduced in [34], nearly matching these lower bounds.

Online service with delay in the clairvoyant setting has also been considered with k servers (rather than a single server). Azar et al. [4] presented a $O(k \cdot \text{polylog}(n))$-competitive randomized algorithm for the problem. The algorithm involved randomly embedding the metric space into a tree, then solving deterministically on that tree; as the algorithm did not use randomization in server allocation, its dependence on k is linear. Subsequently, for the special case of a metric space that is a weighted star, Gupta et al. [26] presented an $O(\log n \log k)$-competitive randomized algorithm. For a general metric space, Gupta et al. [25] presented a $O(\text{polylog}(\Delta, n))$-competitive algorithm, where Δ is the aspect ratio of the metric space (largest-to-smallest pairwise-distance ratio). For the special case of a uniform

metric space, Krnetic et al. [30] settled the exact competitive ratio admitted by the problem; interestingly, the upper bound was achieved by a *non-clairvoyant* algorithm, showing that clairvoyance is not needed on a uniform space.

Another noteworthy line of work in online algorithms with delay has been on online matching, where matching requests arrive over time and accumulate delay cost while unmatched; see, e.g., [1, 22, 3, 12, 13, 5, 32, 27, 20]. In most of these works, a main assumption is that delay accumulation is identical for all requests, which nullifies the need for clairvoyance. (A notable exception is [20], that considers more general delay models.)

1.3 Our Techniques

Consider the algorithm for clairvoyant online service with delay in [35]. This algorithm performs *services*; a service is a sequence of server movements that occurs instantaneously at a given time. Each service is triggered by a set of requests gathering sufficient delay cost. In each service, the server moves within a ball of a certain radius around its location, serving some pending requests subject to a given budget; the radius of the ball and the service budget are determined by a property called the service's *level*.

The prioritization of requests within the ball is according to the future delay accumulation of requests. Intuitively, this prioritization ensures that if a request "survived" a service λ at time t, but then gathers delay cost that triggers another service λ' at time t', then the requests that *were* served by λ would have gathered large delay during $[t, t']$ had they been left pending; this is used to justify a doubling argument, e.g. allowing λ' to have twice the budget as λ. Services that use this doubling mechanism are called "secondary", while other services are called "primary".

However, in the non-clairvoyant setting considered in this paper, we cannot predict future delay, and thus cannot prioritize requests. Instead, we are relegated to spending budget "blindly". In our algorithm, this is expressed by solving a prize-collecting Steiner tree problem in which we aim to visit some locations within a certain ball, where the penalty function is uniform; i.e., visiting every location is equally important. This results in our algorithm visiting the most locations subject to a certain budget per location. Conversely – and crucially – we maintain the property that locations *not* visited by our solution are expensive to visit; in fact, every *bundle* of these requests is expensive. The construction of a poly-time prize-collecting algorithm with this property is based on the construction in [34], and hinges on prize-collecting Steiner tree admitting a Lagrangian approximation (as shown, e.g., by Goemans and Williamson [23]).

Then, we wait for these unvisited locations to gather delay equal to their budget; once enough locations accumulate this delay, we can charge it to the optimal solution, as both the incurred delay and the cost of serving these requests are large. However, when unvisited locations gather large delay, the algorithm must also serve them; it does so greedily, moving to the request and back. We call these greedy services "tertiary", and they exist alongside primary and secondary services.

An additional, more technical component of the algorithm is that of *domes*. As in [35], requests have levels in our algorithm, and a service starts when requests of level at most ℓ gather enough delay in a radius-2^ℓ ball, for some level ℓ. However, unlike in [35], inside an inner ball of radius $2^{\ell-1}$, we only consider delay of requests *exactly* ℓ, forming a dome-like structure. The benefit of this structure is in eliminating the slack in service levels that existed in the doubling argument of [35]. The use of this property in the proof is related to the investment counters that are maintained per location, rather than per request (as in [35]), as non-clairvoyance prohibits us from knowing a request's future delay cost.

2 Preliminaries

In online service with delay, a metric space G of n points is given; we denote by $\delta(u,v)$ the distance between two points $u,v \in G$. Requests are released over time in an online manner, where every request q is associated with a point $V(q) \in G$. At any point in time, the algorithm can move the server from its current location a to a new location a', paying a cost of $\delta(a,a')$. Then, every request pending q where $V(q) = a'$ becomes served. We denote by r_q the release time of request q. Every request q also has some continuous, non-decreasing delay function d_q, which maps from a time $t \geq r_q$ to the delay cost incurred by q if pending until time t[1]; we assume that every request must eventually be served, and thus d_q tends to infinity as time advances. Without loss of generality, we also assume that $d_q(r_q) = 0$ (assuming otherwise would simply add a constant additive term to the cost of all solutions). We expand this notation to multiple requests: for every subset of requests $Q' \subseteq Q$ and time t, we define $d_{Q'}(t) := \sum_{q \in Q'} d_q(t)$. We also define $m := |Q|$ to be the total number of requests in the input sequence.

Our algorithms perform *services*, which are a set of server movements that take place instantaneously. Where λ is a service, we denote by t_λ the time in which the service occurs. We define $a(t), a^*(t)$ to be the locations of the algorithm's server and the optimum's server at time t, respectively. Throughout the paper, when we refer to the value of some variable at t_λ for some service λ, we refer to the value immediately *before* the service. For example, $a(t_\lambda)$ is the location of the server immediately before λ. As a shorthand, we also define $a_\lambda := a(t_\lambda)$.

We define $B(v,r)$ to be the closed ball centered at v of radius r (i.e., the set of points of distance at most r from v). We also sometimes equate the points of the metric space G with the nodes of a clique graph, where the edge connecting nodes u,v has weight $\delta(u,v)$. This graph representation is useful when discussing charging cylinders, a useful tool in our analysis.

For ease of notation, we use the superscript $+$ to indicate the positive part of a number. That is, $x^+ := \max(0,x)$.

3 Online Service with Delay

This section introduces an algorithm for non-clairvoyant online service with delay, proving Theorem 1. First, we define k to be the number of locations on which requests are released in the input. Note that $k \leq \min(n,m)$; we thus focus on proving $O(\sqrt{k}\log k)$-competitiveness, which implies the competitiveness bound of Theorem 1.

3.1 The Algorithm

Before describing the algorithm, we first introduce some of its components.

Steiner tree. As part of our algorithm, we formulate and solve a Steiner tree problem. In Steiner tree, one is given a graph with edge costs and a set of terminal nodes. The goal is to buy an edge subset of minimal cost that connects the terminals. With terminals V, and assuming that the graph is known from context, we use $\mathrm{ST}^*(V)$ to denote the optimal cost of a solution. We also sometimes refer to the (equivalent) rooted version, in which the terminals must be connected to a designated root node r; here, we use $\mathrm{ST}^*(V;r)$ to denote the optimal cost.

[1] The continuity assumption is without loss of generality: relaxing this assumption, one could simulate continuous delay growth upon observing a delay "spike".

Prize-collecting Steiner tree and Lagrangian approximation. In the prize-collecting variant of the (rooted) Steiner tree problem, we are also given a penalty function π mapping from each terminal to a non-negative penalty; we write the prize-collecting input as $(Q, \pi; r)$. The algorithm must connect terminals to the root by buying edges, and must pay the penalty for unconnected terminals. The goal of the algorithm is to minimize the sum of edge costs and penalty costs; we write $\text{PCST}^*(Q, \pi; r)$ to refer to the optimal solution to the input $(Q, \pi; r)$. A *Lagrangian* prize-collecting algorithm is an algorithm that has different approximation ratios w.r.t. these two costs, such that it is 1-approximate on penalty costs. Specifically, Goemans and Williamson [23] presented the following algorithm for prize-collecting Steiner tree.

▶ **Theorem 3** (due to [23]). *There exists an approximation algorithm* PCST *for prize-collecting Steiner tree such that, fixing any input* $(Q, \pi; r)$*, it holds that*

$$\text{PCST}^b(Q, \pi; r) + 2 \cdot \text{PCST}^p(Q, \pi; r) \leq 2 \cdot \text{PCST}^*(Q, \pi; r),$$

where $\text{PCST}^b, \text{PCST}^p$ *are the buying and penalty costs of* PCST*, respectively.*

In [34], a simple procedure is presented that converts a Lagrangian prize-collecting procedure into a procedure which identifies a solution to a maximal subset of requests subject to a budget per request. Combining this result with the algorithm of [23] yields Lemma 4, which introduces the procedure PCSolve used in our algorithm.

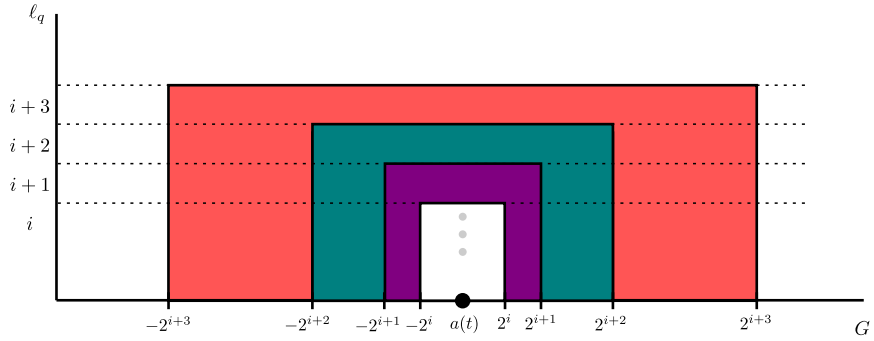
▶ **Lemma 4** (due to [23] and Proposition 21 from [34]). *There exists a procedure* PCSolve *which, given a prize-collecting Steiner tree input* $(Q, \pi; r)$*, outputs a solution* T *for a request subset* $Q' \subseteq Q$ *such that:*

- $c(T) \leq 2 \cdot \sum_{q \in Q'} \pi(q)$.
- *For every* $Q'' \subseteq Q \setminus Q'$*, it holds that* $\text{ST}^*(Q''; r) \geq \sum_{q \in Q''} \pi(q)$.

Levels and pointers. The algorithm maintains for every pending request q a level ℓ_q, which is initially $-\infty$. A service λ also has a level ℓ_λ, which determines its budget for serving requests. As services occur, they may increase the levels of requests. The algorithm also maintains for request q a pointer μ_q to the last service that increased the level of q. (A fine point, as seen in the algorithm, is that a service may also appear in the pointer μ_q only for "attempting" to increase ℓ_q.) Finally, overloading notation, each service λ also has a pointer μ_λ that points to a prior service; this pointer is equal to the pointer μ_q for some request q considered by λ.

Residual delay counters and domes. The algorithm maintains a *residual delay* counter $g_{v,\ell}$ for every location v and level ℓ; for time t, we define $g_{v,\ell}(t)$ to be the value of $g_{v,\ell}$ at time t. This counter grows with the delay cost incurred by level-ℓ requests on v. In addition, the counters are occasionally decreased by services made by the algorithm; this can be interpreted as the service "paying" for incurred delay out of its budget. Positive counters correspond to incurred delay that has not yet been handled; such counters can trigger a service, in the manner we now describe.

At any time t, and for every level ℓ, the algorithm considers positive residual delay counters of levels at most ℓ inside $B\big(a(t), 2^\ell\big)$. Specifically, it considers the total unhandled residual delay of requests of level at most ℓ inside the ring $B\big(a(t), 2^\ell\big) \setminus B\big(a(t), 2^{\ell-1}\big)$, plus unhandled residual delay of requests of level *exactly* ℓ inside the internal ball $B\big(a(t), 2^{\ell-1}\big)$. This forms a dome-like structure, as shown in Figure 1. The algorithm waits until a dome corresponding to some level ℓ becomes critical, i.e., has a lot of unhandled residual delay, and then starts a service. These notions are formalized in Definition 5.



This visualization illustrates the structure of domes. The metric space visualized is a line, shown on the horizontal axis. The vertical axis shows counter levels. The shapes in color correspond to the different domes at the current time, which include different levels at different points, according to their distance from the server.

Figure 1 Visualization of domes.

▶ **Definition 5** (domes). *For every time t, level ℓ, and $v \in B\big(a(t), 2^{\ell}\big)$, define*

$$y_\ell(v,t) := \begin{cases} (g_{v,\ell}(t))^+ & v \in B\big(a(t), 2^{\ell-1}\big) \\ \sum_{\ell' \le \ell} (g_{v,\ell'}(t))^+ & v \in B\big(a(t), 2^{\ell}\big) \setminus B\big(a(t), 2^{\ell-1}\big) \end{cases}$$

For time t and level ℓ, define $Y_\ell(t) := \sum_{v \in B(a(t),2^\ell)} y_\ell(v,t)$; where t is known from context, we also write Y_ℓ. If $Y_\ell \ge 2^\ell$, we say that dome ℓ is critical.

Algorithm's Description. Whenever dome ℓ becomes critical, we start a service λ of level $\ell + 4$. (If more than one dome is critical at the same time, we handle the dome of the largest level first.) The service first considers whether the dome has any positive residual delay from the inner ball $B\big(a_\lambda, 2^{\ell-1}\big) = B\big(a_\lambda, 2^{\ell_\lambda - 5}\big)$. If not, the service λ is called *primary*. Otherwise, consider a location v of the inner ball that contributes positive residual delay to the dome; it must be that $g_{v,\ell} > 0$, which implies (through Proposition 9) that there exists a level-ℓ pending request on v. The service arbitrarily chooses such a request σ_λ, and observes the last service to modify the level of the request, i.e., μ_{σ_λ}; denote this service by λ'. Depending on λ', the algorithm decides if the service will invest for the future ("secondary" service) or act greedily ("tertiary" service). Specifically, the algorithm maintains a counter $\beta(\lambda')$ for the number of times λ' has triggered tertiary services; if the counter is low, λ becomes tertiary and increments this counter; otherwise, λ will be secondary. After deciding if a service is primary, secondary, or tertiary, the algorithm zeroes all positive residual delay on locations in $B\big(a_\lambda, 2^{\ell_\lambda}\big)$ of levels up to ℓ_λ; in particular, this zeroes the residual delay that triggered λ.

Next, if the service is tertiary, it simply moves the server to σ_λ and back, concluding the service. Otherwise, λ is primary or secondary, and thus invests in serving pending requests, through the method ServeEligible. In ServeEligible, the service considers the subset of locations V_λ inside $B\big(a_\lambda, 2^{\ell_\lambda}\big)$ on which there are pending requests. To each location in V_λ, the algorithm assigns a budget of $x_\lambda = 2^{\ell_\lambda}/\sqrt{|V_\lambda|}$ to visiting each such location; the locations V_λ, together with their budgets as penalties, are transferred to PCSolve as a prize-collecting Steiner tree input (where the root is the current location of the server a_λ). PCSolve outputs a tree connecting some locations $Q^\circ_\lambda \subseteq V_\lambda$ to a_λ; this tree is then traversed by the server in a DFS fashion, visiting Q°_λ and ending at a_λ. For the remaining locations $V_\lambda \setminus Q^\circ_\lambda$, where

PCSolve pays the penalty, the algorithm decreases the delay counters for those locations by x_λ. On those locations, the algorithm makes pending requests of level at most ℓ_λ point to λ, and upgrades their level to ℓ_λ.

Finally, for primary services, the algorithm may move the server to a new location a'_λ. Recall that λ starts upon dome $\ell_\lambda - 4$ becoming critical; if the majority of the residual delay making dome $\ell_\lambda - 4$ critical occurs inside a small-radius ball (specifically, radius $2^{\ell_\lambda - 8}$), then the center of this ball becomes the new location a'_λ, at which the server rests at the end of the service.

The non-clairvoyant algorithm for online service with delay is given in Algorithm 1, and the ServeEligible method appears in Algorithm 2. We henceforth focus on analyzing Algorithm 1 and proving Theorem 1.

Algorithm 1 Non-clairvoyant algorithm for online service with delay.

Function UponCritical *(ℓ)*
  start a service λ, and set $\ell_\lambda \leftarrow \ell + 4$.
  denote by t the current time, and by a_λ the current location of the server.
  let $V^\dagger_\lambda \subseteq B(a_\lambda, 2^{\ell_\lambda - 4})$ be the subset of locations v where $y_{\ell_\lambda - 4}(v,t) > 0$.
  if $V^\dagger_\lambda \cap B(a_\lambda, 2^{\ell_\lambda - 5}) = \emptyset$ **then**
    say λ is *primary*.
  else
    let σ_λ be an arbitrary request of level $\ell_\lambda - 4$ on a point in $V^\dagger_\lambda \cap B(a_\lambda, 2^{\ell_\lambda - 5})$.
    define $\mu_\lambda \leftarrow \mu_{\sigma_\lambda}$.
    if $\beta(\mu_\lambda) \geq 2\sqrt{|V_{\mu_\lambda}|}$ **then**
      say λ is *secondary*.
    else
      say λ is *tertiary*.
  // if primary and most residual delay is in small-radius ball, mark its center for future movement.
  if λ *is primary* **and** $\exists a' \in G$*:* $y_{\ell_\lambda - 4}\left(B(a', 2^{\ell_\lambda - 8}) \cap V^\dagger_\lambda, t\right) > 2^{\ell_\lambda - 5}$ **then** define $a'_\lambda := a'$.
  foreach $v \in B(a_\lambda, 2^{\ell_\lambda})$ **do** *// zero residual delay inside service ball, of levels at most ℓ_λ.*
    foreach $\ell' \leq \ell_\lambda$ **do**
      set $g_{v,\ell'} \leftarrow \min(g_{v,\ell'}, 0)$
  if λ *is tertiary* **then**
    visit σ_λ with the server, and return to a_λ afterwards†.
    set $\beta(\mu_\lambda) \leftarrow \beta(\mu_\lambda) + 1$.
    return
  call ServeEligible(λ).
  if λ *is primary* **then** move the server from a_λ to a'_λ.
  set $\beta(\lambda) \leftarrow 0$.

† In Line 19 of this algorithm, and in Line 5 of Algorithm 2, to avoid serving requests without accounting for their delay, the algorithm does not consider requests of level greater than ℓ_λ as served by the movements in λ. thus, the algorithm might consider some requests as still pending when they are, in fact, served.

Algorithm 2 The ServeEligible method.

```
1  Function ServeEligible(λ)
2      let V_λ be the subset of locations in B(a_λ, 2^{ℓ_λ}) on which there are pending requests.
3      define x_λ ← 2^{ℓ_λ} / √|V_λ|.
4      run PCSolve(V_λ, x_λ; a_λ) to obtain a solution T to the subset Q°_λ ⊆ V_λ of locations.
5      traverse T with the server using DFS, returning to a_λ at the end; let Q_λ be the set of
         requests of types in Q°_λ that are served†.
       // upgrade surviving eligible requests, and invest in eligible location counters.
6      foreach v ∈ V_λ do
7          foreach request q on v of level at most ℓ_λ do
8              set ℓ_q ← ℓ_λ.
9              set μ_q ← λ.
10         decrease g_{v,ℓ_λ} by x_λ.
```

3.2 Definitions and Properties

▶ **Definition 6.** *We define the following.*

1. *Let Λ denote the set of services in the algorithm. We partition Λ into $\Lambda^{\mathbf{1}}, \Lambda^{\mathbf{2}}, \Lambda^{\mathbf{3}}$, the sets of primary, secondary, and tertiary services, respectively.*
2. *For two services $\lambda_1, \lambda_2 \in \Lambda$ where λ_2 occurs after λ_1, if $\mu_{\lambda_2} = \lambda_1$ we say that λ_2 is* assigned *to λ_1. Note that in this case $\lambda_1 \in \Lambda^{\mathbf{1}} \cup \Lambda^{\mathbf{2}}$, $\lambda_2 \in \Lambda^{\mathbf{2}} \cup \Lambda^{\mathbf{3}}$.*
3. *Let $\lambda \in \Lambda^{\mathbf{1}} \cup \Lambda^{\mathbf{2}}$ be a service. If there exists a secondary service $\lambda' \in \Lambda^{\mathbf{2}}$ which is assigned to λ, then we call λ a* certified *service, and say that λ' certified λ. We denote the set of certified services by $\Lambda^{\mathbf{c}} \subseteq \Lambda^{\mathbf{1}} \cup \Lambda^{\mathbf{2}}$.*

▶ **Proposition 7.** *For a certified service $\lambda \in \Lambda^{\mathbf{c}}$, we have the following:*

1. *The service λ is certified by exactly one service $\lambda' \in \Lambda^{\mathbf{2}}$.*
2. *It holds that $\ell_{\lambda'} = \ell_\lambda + 4$.*
3. *It holds that $\delta(a_\lambda, a_{\lambda'}) \leq 2^{\ell_\lambda + 1}$.*

Proof. First, observing Line 9 of Algorithm 2, we note that if for request q we have $\mu_q = \lambda$, then it must be the case that $\ell_q = \ell_\lambda$ at that time. Note that λ is only certified by some $\lambda' \in \Lambda^{\mathbf{2}}$ if $\mu_{\sigma_{\lambda'}} = \lambda$, and thus $\ell_{\sigma_{\lambda'}} = \ell_\lambda$ at $t_{\lambda'}$; but, $\ell_{\lambda'} = \ell_{\sigma_{\lambda'}} + 4$ at $t_{\lambda'}$, proving the second item. Moreover, note that $\sigma_{\lambda'} \in V_\lambda$, and thus $\delta(a_\lambda, \sigma_{\lambda'}) \leq 2^{\ell_\lambda}$. In addition, since $\sigma_{\lambda'} \in B\left(a_{\lambda'}, 2^{\ell_{\lambda'} - 5}\right)$, we have $\delta(a_{\lambda'}, \sigma_{\lambda'}) \leq 2^{\ell_{\lambda'} - 5} = 2^{\ell_\lambda - 1}$. The triangle inequality thus implies $\delta(a_\lambda, a_{\lambda'}) \leq 2^{\ell_\lambda} + 2^{\ell_\lambda - 1} \leq 2^{\ell_\lambda + 1}$, proving the third item.

We now prove the first item, i.e., the uniqueness of λ'. Suppose, for contradiction, that a certified service $\lambda \in \Lambda^{\mathbf{c}}$ is certified by two services $\lambda', \lambda'' \in \Lambda^{\mathbf{2}}$, and assume without loss of generality that λ' occurs before λ''. We prove that after λ', there remains no pending request q with $\mu_q = \lambda$, and thus λ'' cannot certify λ later on. Indeed, consider such a request q immediately before λ'; it holds that $q \in V_\lambda \subseteq B\left(a_\lambda, 2^{\ell_\lambda}\right)$, and it must also be the case that $\ell_q = \ell_\lambda$ at that time. But, we've shown that $\delta(a_\lambda, a_{\lambda'}) \leq 2^{\ell_\lambda + 1}$, and thus $B\left(a_\lambda, 2^{\ell_\lambda}\right) \subseteq B\left(a_{\lambda'}, 2^{\ell_\lambda} + 4\right) = B\left(a_{\lambda'}, 2^{\ell_{\lambda'}}\right)$. Thus, $q \in V_{\lambda'}$, and will either be served by λ' or have μ_q be set to λ', in contradiction to having $\mu_q = \lambda$ at $t_{\lambda''}$. ◀

▶ **Proposition 8.** *For every level i it holds that $Y_i \leq 2^{i+2}$ at every point in time.*

The proof of Proposition 8 is in Section B.

▶ **Proposition 9.** *For every point $v \in G$, level ℓ and time t, it holds that $g_{v,\ell}(t) \le d_{Q'}(t)$, where Q' is the set of* pending *requests of level exactly ℓ on v at time t. (In particular, when $Q' = \emptyset$, it holds that $g_{v,\ell}(t) \le 0$.)*

Proof. We prove this by induction as time advances, noting that the proposition holds at time 0 (before any requests are released). Note that $g_{v,\ell}(t)$ grows at the same rate as the delay of pending requests of level ℓ on v, and thus delay growth cannot break the invariant. In addition, counter decreases in services cannot break the invariant. The only risky event is when a request of level ℓ is upgraded or completed. But, this only happens in a service λ such that $\ell_\lambda \ge \ell$ and $v \in B\left(a_\lambda, 2^{\ell_\lambda}\right)$. Note that in such services, Line 17 of Algorithm 1 ensures that $g_{v,\ell}$ is non-positive after the service, maintaining the invariant. ◀

▶ **Proposition 10.** *During a service λ, if the algorithm moves its server from a_λ to a'_λ in Line 23 of Algorithm 1, then $2^{\ell_\lambda - 5} - 2^{\ell_\lambda - 8} \le \delta\left(a_\lambda, a'_\lambda\right) \le 2^{\ell_\lambda - 4} + 2^{\ell_\lambda - 8}$.*

Proof. Note that the server only moves to a'_λ when λ is primary. In this case, it holds that $V^\dagger_\lambda \cap B\left(a_\lambda, 2^{\ell_\lambda - 5}\right) = \emptyset$. But, due to Proposition 9, this implies that $g_{v,\ell_\lambda - 4}(t) \le 0$ for every $v \in B\left(a_\lambda, 2^{\ell_\lambda - 5}\right)$; thus, for such v we have $y_{\ell_\lambda - 4}(v,t) = 0$, and thus $V^\dagger_\lambda$ is contained in the ring $B\left(a_\lambda, 2^{\ell_\lambda - 4}\right) \setminus B\left(a_\lambda, 2^{\ell_\lambda - 5}\right)$. But, from the definition of a'_λ, $B\left(a'_\lambda, 2^{\ell_\lambda - 8}\right)$ must contain a point from $V^\dagger_\lambda$, which completes the proof. ◀

For a service λ, define the cost of the service, denoted $c(\lambda)$, to be the total movement of the server during λ plus the total amount by which the service decreases counters $g_{v,\ell}$.

▶ **Proposition 11.** $\mathrm{ALG} \le \sum_{\lambda \in \Lambda} c(\lambda)$.

Proof. To prove the proposition, we just have to prove that the total delay incurred by the algorithm is upper-bounded by the total amount by which counters are decreased in the algorithm. First, recall the assumption that the delay of requests tends to infinity as time advances; thus, every request is eventually served by the algorithm. Thus, once all requests are served, Proposition 9 implies that $g_{v,\ell} \le 0$ for every $v \in G$ and level ℓ, which completes the proof. ◀

The following observation results from the fact that every counter $g_{v,\ell}$ is counted in exactly one dome (and can also easily be seen in Figure 1).

▶ **Observation 12.** *At any time t, and for every level ℓ, it holds that*

$$\sum_{\ell' \le \ell} Y_{\ell'} = \sum_{v \in B\left(a(t), 2^\ell\right)} \sum_{\ell' \le \ell} \left(g_{v,\ell'}\right)^+.$$

▶ **Lemma 13.** $\mathrm{ALG} \le O(\sqrt{k}) \cdot \sum_{\lambda \in \Lambda^{\mathbf{1}}} 2^{\ell_\lambda} + O(\sqrt{k}) \cdot \sum_{\lambda \in \Lambda^{\mathbf{e}}} 2^{\ell_\lambda}$.

Proof. Applying Proposition 11, it is enough to bound $\sum_{\lambda \in \Lambda} c(\lambda)$. We first bound the cost of tertiary services $\sum_{\lambda \in \Lambda^{\mathbf{3}}} c(\lambda)$. Consider a tertiary service $\lambda \in \Lambda^{\mathbf{3}}$; it incurs the following costs:

1. It zeroes any positive counters $g_{v,\ell}$ for every $v \in B\left(a_\lambda, 2^{\ell_\lambda}\right)$ and $\ell \le \ell_\lambda$. The cost of this is at most

$$\sum_{v \in B\left(a_\lambda, 2^{\ell_\lambda}\right)} \sum_{\ell \le \ell_\lambda} \left(g_{v,\ell}\right)^+ = \sum_{\ell \le \ell_\lambda} Y_\ell \le 2^{\ell_\lambda + 3},$$

where the equality is due to Observation 12, and the inequality is due to Proposition 8 and summing a geometric sequence.

2. It moves the server from a_λ to σ_λ and back (Line 19 of Algorithm 1). Since $\sigma_\lambda \in B(a_\lambda, 2^{\ell_\lambda - 4})$, the cost of this is at most $2^{\ell_\lambda - 3}$.

Overall, the cost of a tertiary service λ is $O(2^{\ell_\lambda})$.

Now, consider a *non-tertiary* service $\lambda \in \Lambda^{\mathbf{1}} \cup \Lambda^{\mathbf{2}}$; there are at most $2 \cdot \sqrt{|V_\lambda|} + 1$ tertiary services assigned to λ', as each such service raises $\beta(\lambda)$ by 1 and $\beta(\lambda)$ does not exceed $2 \cdot \sqrt{|V_\lambda|} + 1$; note that $2 \cdot \sqrt{|V_\lambda|} + 1 = O(\sqrt{k})$. Moreover, for a tertiary service λ' assigned to λ we have $\ell_{\lambda'} = \ell_\lambda + 4$. Overall, we have that the cost of tertiary services is $O(\sqrt{k}) \cdot \sum_{\lambda \in \Lambda^{\mathbf{1}} \cup \Lambda^{\mathbf{2}}} 2^{\ell_\lambda}$.

Next, we bound the cost of non-tertiary services. The cost of a service $\lambda \in \Lambda^{\mathbf{1}} \cup \Lambda^{\mathbf{2}}$ consists of the following terms:

1. It zeroes positive counters $g_{v,\ell}$ for every $v \in B(a_\lambda, 2^{\ell_\lambda})$ and $\ell \le \ell_\lambda$. This cost can be bounded by $O(1) \cdot 2^{\ell_\lambda}$, using the same argument as for tertiary services.
2. In Algorithm 2, the algorithm moves the server (Line 5) and decreases counters (Line 10). From the guarantee of PCSolve in Lemma 4, the cost of moving the server is at most $2 \cdot 2 \cdot x_\lambda \cdot |Q^\circ_\lambda|$; the cost of decreasing counters is $(|V_\lambda| - |Q^\circ_\lambda|) \cdot x_\lambda$. Overall, the cost of this item is at most $O(1) \cdot |V_\lambda| \cdot x_\lambda$, which is at most $O(1) \cdot 2^{\ell_\lambda} \cdot \sqrt{|V_\lambda|}$. Using the fact that $|V_\lambda| \le k$, the cost of this item is at most $O(\sqrt{k}) \cdot 2^{\ell_\lambda}$.
3. In Line 23 of Algorithm 1, the server might move to a new location a'_λ. However, $\delta(a_\lambda, a'_\lambda) \le 2^{\ell_\lambda - 4} + 2^{\ell_\lambda - 8}$ due to Proposition 10. Thus, the cost of this movement is also $O(1) \cdot 2^{\ell_\lambda}$.

Overall, it holds that $c(\lambda) \le O(\sqrt{k}) \cdot 2^{\ell_\lambda}$; combining this with the bound for tertiary services, we get

$$\mathrm{ALG} \le O(\sqrt{k}) \cdot \sum_{\lambda \in \Lambda^{\mathbf{1}} \cup \Lambda^{\mathbf{2}}} 2^{\ell_\lambda}. \tag{1}$$

Finally, we bound $\sum_{\lambda \in \Lambda^{\mathbf{2}}} 2^{\ell_\lambda} \le O(1) \sum_{\lambda \in \Lambda^{\mathbf{c}}} 2^{\ell_\lambda}$, which completes the proof. Consider a secondary service $\lambda \in \Lambda^{\mathbf{2}}$, which certifies some prior service $\lambda' = \mu_\lambda$. Through Proposition 7, it holds that $\ell_{\lambda'} = \ell_\lambda - 4$, and thus $2^{\ell_\lambda} \le O(1) \cdot 2^{\ell_{\lambda'}}$. Again through Proposition 7, it holds that λ' is only certified once (by λ); thus, summing over secondary services yields

$$\sum_{\lambda \in \Lambda^{\mathbf{2}}} 2^{\ell_\lambda} \le O(1) \cdot \sum_{\lambda \in \Lambda^{\mathbf{1}}} 2^{\ell_\lambda},$$

which combined with Equation (1) completes the proof. ◀

3.3 Charging Cylinders

It remains to bound the terms $\sum_{\lambda \in \Lambda^{\mathbf{1}}} 2^{\ell_\lambda}$ and $\sum_{\lambda \in \Lambda^{\mathbf{c}}} 2^{\ell_\lambda}$. To do so, we restate the notions of charging balls and charging cylinders introduced in [35].

First, consider our goal. The optimal solution takes some tour through the graph over time, and we would like to map its resulting cost to the services of the algorithm. To this end, we give each service λ temporary "ownership" of a set of edges and edge parts, for some interval in time. We then charge the cost associated with λ (i.e. 2^{ℓ_λ}) to the total cost incurred by the optimal solution in traversing these edges and edge parts during the given interval. In doing so, one must ensure that no movement of OPT is charged to twice; this is ensured by having *disjointness*, which is the property that no edge part is owned by more than one service at any point in time.

Charging shapes and cylinders. A *charging shape* is a shape in the metric space that contains some edges and "parts" of edges. (For the sake of this definition, an edge e can be partitioned into parts whose weights sum up to $c(e)$.)

A *charging ball* centered at v of radius r – which, overloading notation, we denote $B(v,r)$ – is a charging shape such that:

- If both u, w are in $B(v,r)$ then $B(v,r)$ contains the entire edge $\{u,w\}$.
- If $u \in B(v,r)$ but $w \notin B(v,r)$, then $B(v,r)$ contains the part of $\{u,w\}$ connected to u of weight $r - \delta(v,u)$.

A *charging cylinder* is a pair (B, I) where I is a time interval and B is a charging shape.

Disjointness. We say that a set of charging shapes is disjoint if the parts of edges that appear in different charging shapes do not intersect. We say that a set of charging cylinders is disjoint if for every time t it holds that the charging shapes of the cylinders whose time intervals contain t are disjoint.

Intersections. For a set of edges $E' \subseteq E$ and charging shape B, we define $c(E' \cap B)$ to be the total weight of edges in E' that appear in B. For a cylinder $\gamma = (B, I)$, we define $c(\mathrm{OPT} \cap \gamma)$ (called the *intersection* of OPT with γ) to be the total weight of edges in E' that appear in B, where E' is the set of edges traversed by OPT at least once during I.

Theorem 14 is due to [35], and is used to relate the total intersection of the cylinder set we construct to the cost of the optimal solution. The intuition for the theorem is the following: suppose we are given a set of cylinders Γ whose shapes are balls centered at N points. One can replace every such charging ball of radius r with a *perforated* charging ball, from which balls of small radius $\Theta(r/N^2)$ are removed around each of the N points; let Γ' be the set of cylinders after this perforation process. In Γ', two cylinders whose shapes are (perforated) charging balls with radii r_1, r_2 where $r_1 \le O(r_2/N^2)$ are guaranteed to be disjoint. Suppose that in the original set Γ, every two cylinders with radii within a constant factor from each other are also disjoint; after the perforation, we can partition Γ' into $\Theta(\log N)$ disjoint classes. This bounds the intersection of Γ' with OPT by at most $\Theta(\log N) \cdot \mathrm{OPT}$. Moreover, one can observe that this perforation of a cylinder's shape decreases its intersection with OPT by at most a constant times the charging ball's radius; this relates the intersection of Γ with OPT to the intersection of Γ' with OPT. For more intuition regarding cylinders and Theorem 14, see Figure 2.

▶ **Theorem 14** ([35]). *Let Γ be a set of cylinders such that for every cylinder $\gamma \in \Gamma$ its charging shape $B(v_\gamma, r_\gamma)$ is a ball. Partition Γ into $\{\Gamma_i\}$ where for every $\gamma \in \Gamma_i$ it holds that $\lceil \log r_\gamma \rceil = i$. If for every i the set of cylinders Γ_i is disjoint, then for every constant $c > 0$ it holds that*

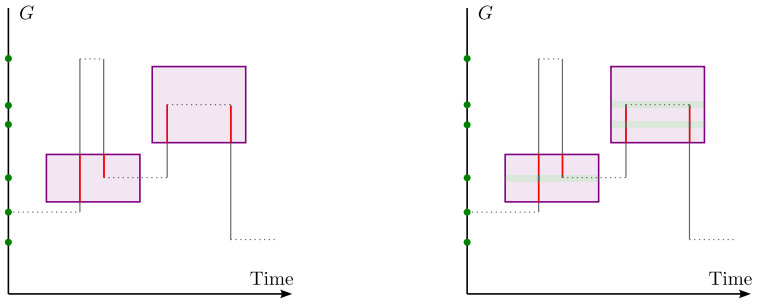
$$\sum_{\gamma \in \Gamma} c(\mathrm{OPT} \cap \gamma) \le O(\log k) \cdot \mathrm{OPT} + c \cdot \sum_{\gamma \in \Gamma} r_\gamma,$$

where k is the number of centers used by cylinders in Γ.

Through constructing and analyzing such cylinders, we prove the following lemmas.

▶ **Lemma 15.** *It holds that $\sum_{\lambda \in \Lambda^I} 2^{\ell_\lambda} \le O(\log k) \cdot \mathrm{OPT}$.*

▶ **Lemma 16.** *It holds that $\sum_{\lambda \in \Lambda^c} 2^{\ell_\lambda} \le O(\log k) \cdot \mathrm{OPT}$.*



(a) cylinders. **(b)** perforated cylinders.

This figure illustrates the concept of cylinders used in this paper and in the proof of Theorem 14. Figure 2a shows a metric space G which is a line, the points of which appear on the vertical axis. The tour of the optimal server appears as an axis-aligned curve traversing space over time; the length of the solid parts of the curve correspond to the movement cost of the optimal solution. Two charging cylinders appear in the figure, whose charging shapes are balls; those balls appear as intervals in the one-dimensional metric space, and thus the cylinder appears as a rectangle in the figure. For each cylinder γ, the part of the optimal tour counted towards $c(\mathrm{OPT} \cap \gamma)$ is shown in red. Note that as the shown cylinders are disjoint, the sum of $\sum_\gamma c(\mathrm{OPT} \cap \gamma)$ lower-bounds OPT. Figure 2b shows the main tool used in proving Theorem 14 in [35], which is perforation; in essence, lower-radii charging balls are removed from each cylinder around each point in the metric space, enabling immediate disjointness with all cylinders of much lower radii.

Figure 2 Illustration of cylinders.

The proof of Lemma 15 appears in Section A, while the proof of Lemma 16 appears in the remainder of this section. We now note that given these lemmas, the proof of the main theorem is complete.

Proof of Theorem 1. The proof is immediate from Lemma 13, Lemma 15 and Lemma 16. ◀

The remainder of this section proves Lemma 16. For every service $\lambda \in \Lambda^{\mathbf{c}}$, we denote by Λ_λ the set of tertiary services assigned to λ. We also define $\Sigma_\lambda := \{\sigma_{\lambda'} | \lambda' \in \Lambda_\lambda\}$; note that $\Sigma_\lambda \subseteq E_\lambda$. Finally, we define $\Sigma^\circ_\lambda := V(\Sigma_\lambda)$.

First, we study the size of the set Σ°_λ.

▶ **Proposition 17.** *For every $\lambda \in \Lambda^{\mathbf{c}}$, it holds that $\left|\Sigma^\circ_\lambda\right| = \left\lceil 2\sqrt{|V_\lambda|} \right\rceil$.*

Proof. First, note that $|\Sigma_\lambda|$ is exactly the final value of the counter $\beta(\lambda)$, as each tertiary service assigned to λ increases $\beta(\lambda)$ by 1 (Line 20 of Algorithm 1). A service assigned to λ will only be tertiary if $\beta(\lambda) < 2\sqrt{|V_\lambda|}$; otherwise, it would be secondary. Thus, the final value of the counter is at most $\left\lceil 2\sqrt{|V_\lambda|} \right\rceil$. However, we know that a secondary service is assigned to λ, as $\lambda \in \Lambda^{\mathbf{c}}$. Thus, the final value of $\beta(\lambda)$ is exactly $\left\lceil 2\sqrt{|V_\lambda|} \right\rceil$, completing the proof. ◀

▶ **Definition 18.** *For a service $\lambda \in \Lambda^{\mathbf{c}}$, define:*

- *E_λ to be the set of pending requests of level at most ℓ_λ on points in Σ°_λ at t_λ.*
- *The certified time interval $I_{\mathrm{c}}(\lambda) := (\min_{q \in E_\lambda} r_q, \max_{\lambda' \in \Lambda_\lambda} t_{\lambda'}]$.*
- *The certified cylinder $\gamma_{\mathrm{c}}(\lambda) := (B(a_\lambda, 3 \cdot 2^{\ell_\lambda}), I_{\mathrm{c}}(\lambda))$.*

We also define the justified residual delay *of λ, which is the amount $D^*_{\mathrm{c},\lambda} := \sum_{v \in V^*_\lambda} x_\lambda$, where $V^*_\lambda \subseteq \Sigma^\circ_\lambda$ is the subset of locations which were not visited by the optimal solution during $I_{\mathrm{c}}(\lambda)$.*

To prove that the intersection of a certified cylinder with the optimal solution is sufficiently large, we first restate a proposition from [35]. Intuitively, it states that if the set of requested terminals in a Steiner tree input is concentrated in a ball, a constant fraction of the cost required for connecting them must be incurred within a padded version of the ball, whose radius is larger by a constant factor.

▶ **Proposition 19** (restatement of Proposition 3.22 from [35]). *Let V be a set of locations that are contained in $B(v,r)$, for some location v and radius r, and let $G' \subseteq G$ be any subgraph connecting V. Then it holds that $c(G' \cap \mathcal{B}(v,3r)) \geq \mathrm{ST}(V)/2$.*

▶ **Proposition 20.** *It holds that $\sum_{\lambda \in \Lambda^e} D^*_{c,\lambda} \leq \mathrm{OPT}$.*

Proof. We observe charging triplets of the form (v,ℓ,I), where v is a location, ℓ is a level, and I is a time interval. Every certified service $\lambda \in \Lambda^{\mathbf{c}}$ *owns* triplets. Specifically, for every service $\lambda \in \Lambda^{\mathbf{c}}$, location $v \in V^*_\lambda$, and λ_v the tertiary service assigned to λ where $\sigma_{\lambda_v} \in v$, we say that λ owns the triplet (v,ℓ_λ,I_v), where $I_v = [t_\lambda, t_{\lambda_v}]$ be the time interval between λ and the service λ_v. We now claim the following:

1. If λ owns triplet v,ℓ,I, then a delay of at least x_λ is incurred by requests of level ℓ on v during I. Moreover, these requests are pending in the optimal solution during the interval I.
2. No two triplets intersect. That is, there are no two services $\lambda_1,\lambda_2 \in \Lambda^{\mathbf{c}}$ such that λ_1 owns (v,ℓ,I_1), λ_2 owns (v,ℓ,I_2) and $I_1 \cap I_2 \neq \emptyset$.

The first claim implies that the optimal solution incurs a delay cost of $D^*_{c,\lambda}$ per service $\lambda \in \Lambda^{\mathbf{c}}$, and the second claim implies that no delay cost is charged twice. Together, these two claims complete the proof.

Claim 1. Fix service $\lambda \in \Lambda^{\mathbf{c}}$, location $v \in V^*_\lambda$, and tertiary service λ_v assigned to λ where $\sigma_{\lambda_v} \in v$; let $I_v = [t_\lambda, t_{\lambda_v}]$, and set $\ell := \ell_\lambda$. As $v \in \Sigma^\circ_\lambda \subseteq V_\lambda$, it holds immediately after λ that $g_{v,\ell_\lambda} \leq -x_\lambda$ (due to Line 17 of Algorithm 1 and Line 10 of Algorithm 2). However, at t_{λ_v}, it holds that $y_{\ell_{\lambda_v}-4}(v) > 0$; moreover, it holds that $v \in B\big(a_{\lambda_v}, 2^{\ell_{\lambda_v}-5}\big)$, and thus $y_{\ell_{\lambda_v}-4}\big(v,t_{\lambda_v}\big) = (g_{v,\ell_{\lambda_v}-4}(t_{\lambda_v}))^+ = (g_{v,\ell_\lambda}(t_{\lambda_v}))^+$. Thus, the counter g_{v,ℓ_λ} has increased by at least x_λ during the time interval $I_v := (t_\lambda, t_{\lambda_v}]$.

Next, we claim that inside I_v, there was no non-tertiary service λ' of level at least ℓ such that $v \in B\big(a_{\lambda'}, 2^{\ell_\lambda}\big)$. Assume otherwise, and note that σ_{λ_v} is pending on v, and is of level exactly ℓ; thus, after λ', it would either be served or have $\mu_{\sigma_{\lambda_v}}$ be set to λ', in contradiction to λ_v being assigned to λ.

As a result, we claim that every request on v of level ℓ_λ during I_v belongs to E_λ. First, note that immediately after λ, all pending level-ℓ requests on v are in E_λ. Suppose for contradiction that this is broken at some point; that is, a new level-ℓ request joins. This could only happen if its level is upgraded by some non-tertiary service during I_v, but there is no such service due to the previous claim.

Finally, note that the requests of E_λ are pending in the optimal solution during I_v, as the optimal server did not visit v during $I_c(\lambda)$; thus, the optimal solution incurs the same delay cost of at least x_λ.

Claim 2. Assume that two triplets (v,ℓ,I_1), (v,ℓ,I_2) that are owned by services $\lambda_1,\lambda_2 \in \Lambda^{\mathbf{c}}$, are such that $I_1 \cap I_2 \neq \emptyset$; note that $\ell_{\lambda_1} = \ell_{\lambda_2} = \ell$. Assume without loss of generality that λ_1 is the earlier service; then, $t_{\lambda_2} \in I_1$. Note that $v \in B\big(a_{\lambda_2}, 2^{\ell_{\lambda_2}}\big)$; however, this cannot be, at no non-tertiary service λ of level at least ℓ can occur during I_1 such that $v \in B\big(a_\lambda, 2^{\ell_\lambda}\big)$, as seen in the proof of the first claim. ◀

▶ **Proposition 21.** *For every $\lambda \in \Lambda^c$, it holds that $2^{\ell_\lambda - 1} \leq c(\mathrm{OPT} \cap \gamma_c(\lambda)) + D^*_{c,\lambda}$.*

Proof. Consider the set of locations $W := \Sigma^\circ_\lambda \setminus V^*_\lambda$, which is a subset of V_λ. These locations were not visited during λ, as the solution computed by PCSolve in Line 4 of Algorithm 2 does not include them. However, due to the guarantee of PCSolve in Lemma 4, this implies that

$$\mathrm{ST}^*(W; a_\lambda) \geq x_\lambda \cdot |W| = 2^{\ell_\lambda}/\sqrt{|V_\lambda|} \cdot |W|.$$

Noting that $W \subseteq B\big(a_\lambda, 2^{\ell_\lambda}\big)$, we have $\mathrm{ST}^*(W) \geq \mathrm{ST}^*(W; a_\lambda) - 2^{\ell_\lambda}$. Applying Proposition 19, denoting by G^* the subgraph traversed by OPT during $I_c(\lambda)$, it holds that $c\big(G^* \cap B\big(a_\lambda, 3 \cdot 2^{\ell_\lambda}\big)\big) \geq \mathrm{ST}^*(W)/2$. Combining the above, we get:

$$\begin{aligned} c(\mathrm{OPT} \cap \gamma_c(\lambda)) + D^*_{c,\lambda} &\geq \mathrm{ST}^*(W)/2 + \left(\left|\Sigma^\circ_\lambda\right| - |W|\right)x_\lambda \\ &\geq \frac{2^{\ell_\lambda - 1}}{\sqrt{|V_\lambda|}} \cdot |W| - 2^{\ell_\lambda - 1} + \left(\left|\Sigma^\circ_\lambda\right| - |W|\right)\frac{2^{\ell_\lambda}}{\sqrt{|V_\lambda|}} \\ &\geq 2^{\ell_\lambda - 1} \cdot \left(\frac{|W|}{\sqrt{|V_\lambda|}} + \frac{\left|\Sigma^\circ_\lambda\right| - |W|}{\sqrt{|V_\lambda|}} - 1\right) = 2^{\ell_\lambda - 1} \cdot \left(\frac{\left|\Sigma^\circ_\lambda\right|}{\sqrt{|V_\lambda|}} - 1\right) \geq 2^{\ell_\lambda - 1} \end{aligned}$$

where the final inequality is due to the fact that $\left|\Sigma^\circ_\lambda\right| = \left\lceil 2\sqrt{|V_\lambda|}\right\rceil$, due to Proposition 17. ◀

▶ **Proposition 22.** *For every ℓ, the set of cylinders $\{\gamma_c(\lambda) | \lambda \in \Lambda^c \wedge \ell_\lambda = \ell\}$ is disjoint.*

The proof of Proposition 22 appears in Section B.

Proof of Lemma 16. It holds that

$$\sum_{\lambda \in \Lambda^c} 2^{\ell_\lambda} \leq 2 \sum_{\lambda \in \Lambda^c} \left(c(\mathrm{OPT} \cap \gamma_c(\lambda)) + D^*_{c,\lambda}\right) \leq O(\log k) \cdot \mathrm{OPT} + O(1) \cdot \mathrm{OPT} = O(\log k) \cdot \mathrm{OPT}$$

where the first inequality is due to Proposition 21, and the second inequality is due to Proposition 22 and Proposition 20. ◀

4 Conclusions

In this paper, we presented a $O(\min\{\sqrt{n}\log n, \sqrt{m}\log m\})$-competitive algorithm for non-clairvoyant online service with delay, that is deterministic and runs in polynomial time. This nearly matches the lower bounds of $\Omega(\sqrt{n})$ and $\Omega(\sqrt{m})$ on competitiveness that appear in [4].

However, the choice of metric space for the problem greatly affects the achievable competitive ratio for non-clairvoyant service with delay. The square-root lower bounds in [4] are obtained for a uniform metric space; however, for a metric space which is a line, an $O(\log n)$-competitive non-clairvoyant algorithm exists [14]. Given this stark difference in competitive ratio, it would be interesting to identify a salient property of the metric space, and study competitiveness as a function of this property.

References

1 Itai Ashlagi, Yossi Azar, Moses Charikar, Ashish Chiplunkar, Ofir Geri, Haim Kaplan, Rahul M. Makhijani, Yuyi Wang, and Roger Wattenhofer. Min-cost bipartite perfect matching with delays. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques, APPROX/RANDOM 2017, August 16-18, 2017, Berkeley, CA, USA*, pages 1:1–1:20, 2017. doi:10.4230/LIPIcs.APPROX-RANDOM.2017.1.

2 Yossi Azar, Ashish Chiplunkar, Shay Kutten, and Noam Touitou. Set cover with delay - clairvoyance is not required. In Fabrizio Grandoni, Grzegorz Herman, and Peter Sanders, editors, *28th Annual European Symposium on Algorithms, ESA 2020, September 7-9, 2020, Pisa, Italy (Virtual Conference)*, volume 173 of *LIPIcs*, pages 8:1–8:21. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2020. doi:10.4230/LIPIcs.ESA.2020.8.

3 Yossi Azar and Amit Jacob Fanani. Deterministic min-cost matching with delays. In *Approximation and Online Algorithms - 16th International Workshop, WAOA 2018, Helsinki, Finland, August 23-24, 2018, Revised Selected Papers*, pages 21–35, 2018. doi:10.1007/978-3-030-04693-4_2.

4 Yossi Azar, Arun Ganesh, Rong Ge, and Debmalya Panigrahi. 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*, pages 551–563, 2017. doi:10.1145/3055399.3055475.

5 Yossi Azar, Runtian Ren, and Danny Vainstein. The min-cost matching with concave delays problem. In Dániel Marx, editor, *Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms, SODA 2021, Virtual Conference, January 10 - 13, 2021*, pages 301–320. SIAM, 2021. doi:10.1137/1.9781611976465.20.

6 Yossi Azar and Noam Touitou. 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*, pages 60–71, 2019. doi:10.1109/FOCS.2019.00013.

7 Yossi Azar and Noam Touitou. 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*, pages 1368–1379. IEEE, 2020. doi:10.1109/FOCS46700.2020.00129.

8 Marcin Bienkowski, Martin Böhm, Jaroslaw Byrka, Marek Chrobak, Christoph Dürr, Lukáš Folwarczný, Lukasz Jez, Jirí Sgall, Kim Thang Nguyen, and Pavel Veselý. New results on multi-level aggregation. *Theor. Comput. Sci.*, 861:133–143, 2021. doi:10.1016/j.tcs.2021.02.016.

9 Marcin Bienkowski, Martin Böhm, Jaroslaw Byrka, Marek Chrobak, Christoph Dürr, Lukáš Folwarczný, Lukasz Jez, Jiri Sgall, Nguyen Kim Thang, and Pavel Veselý. Online algorithms for multi-level aggregation. In *24th Annual European Symposium on Algorithms, ESA 2016, August 22-24, 2016, Aarhus, Denmark*, pages 12:1–12:17, 2016. doi:10.4230/LIPIcs.ESA.2016.12.

10 Marcin Bienkowski, Martin Böhm, Jaroslaw Byrka, and Jan Marcinkowski. Online facility location with linear delay. In Amit Chakrabarti and Chaitanya Swamy, editors, *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques, APPROX/RANDOM 2022, September 19-21, 2022, University of Illinois, Urbana-Champaign, USA (Virtual Conference)*, volume 245 of *LIPIcs*, pages 45:1–45:17. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2022. doi:10.4230/LIPIcs.APPROX/RANDOM.2022.45.

11 Marcin Bienkowski, Jaroslaw Byrka, Marek Chrobak, Lukasz Jez, Dorian Nogneng, and Jirí Sgall. 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*, pages 42–54, 2014. doi:10.1137/1.9781611973402.4.

12 Marcin Bienkowski, Artur Kraska, Hsiang-Hsuan Liu, and Pawel Schmidt. A primal-dual online deterministic algorithm for matching with delays. In *Approximation and Online Algorithms - 16th International Workshop, WAOA 2018, Helsinki, Finland, August 23-24, 2018, Revised Selected Papers*, pages 51–68, 2018. doi:10.1007/978-3-030-04693-4_4.

13 Marcin Bienkowski, Artur Kraska, and Pawel Schmidt. A match in time saves nine: Deterministic online matching with delays. In *Approximation and Online Algorithms - 15th International Workshop, WAOA 2017, Vienna, Austria, September 7-8, 2017, Revised Selected Papers*, pages 132–146, 2017. doi:10.1007/978-3-319-89441-6_11.

14 Marcin Bienkowski, Artur Kraska, and Pawel Schmidt. 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*, pages 237–248, 2018. doi:10.1007/978-3-030-01325-7_22.

15 Carlos Fisch Brito, Elias Koutsoupias, and Shailesh Vaya. Competitive analysis of organization networks or multicast acknowledgment: How much to wait? *Algorithmica*, 64(4):584–605, 2012. doi:10.1007/s00453-011-9567-5.

16 Niv Buchbinder, Moran Feldman, Joseph (Seffi) Naor, and Ohad Talmon. 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*, pages 1235–1244, 2017. doi:10.1137/1.9781611974782.80.

17 Niv Buchbinder, Kamal Jain, and Joseph Naor. Online primal-dual algorithms for maximizing ad-auctions revenue. In *Algorithms - ESA 2007, 15th Annual European Symposium, Eilat, Israel, October 8-10, 2007, Proceedings*, pages 253–264, 2007. doi:10.1007/978-3-540-75520-3_24.

18 Niv Buchbinder, Tracy Kimbrel, Retsef Levi, Konstantin Makarychev, and Maxim Sviridenko. 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*, pages 952–961, 2008. URL: http://dl.acm.org/citation.cfm?id=1347082.1347186.

19 Ryder Chen, Jahanvi Khatkar, and Seeun William Umboh. Online weighted cardinality joint replenishment problem with delay. In Mikolaj Bojanczyk, Emanuela Merelli, and David P. Woodruff, editors, *49th International Colloquium on Automata, Languages, and Programming, ICALP 2022, July 4-8, 2022, Paris, France*, volume 229 of *LIPIcs*, pages 40:1–40:18. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2022. doi:10.4230/LIPIcs.ICALP.2022.40.

20 Lindsey Deryckere and Seeun William Umboh. Online matching with set and concave delays. In Nicole Megow and Adam D. Smith, editors, *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques, APPROX/RANDOM 2023, September 11-13, 2023, Atlanta, Georgia, USA*, volume 275 of *LIPIcs*, pages 17:1–17:17. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2023. doi:10.4230/LIPICS.APPROX/RANDOM.2023.17.

21 Daniel R. Dooly, Sally A. Goldman, and Stephen D. Scott. 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*, pages 389–398, 1998. doi:10.1145/276698.276792.

22 Yuval Emek, Yaacov Shapiro, and Yuyi Wang. Minimum cost perfect matching with delays for two sources. In *Algorithms and Complexity - 10th International Conference, CIAC 2017, Athens, Greece, May 24-26, 2017, Proceedings*, pages 209–221, 2017. doi:10.1007/978-3-319-57586-5_18.

23 Michel X. Goemans and David P. Williamson. A general approximation technique for constrained forest problems. In *Proceedings of the Third Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '92, pages 307–316, Philadelphia, PA, USA, 1992. Society for Industrial and Applied Mathematics. URL: http://dl.acm.org/citation.cfm?id=139404.139468.

24 Anupam Gupta, Amit Kumar, and Debmalya Panigrahi. Caching with time windows. In Konstantin Makarychev, Yury Makarychev, Madhur Tulsiani, Gautam Kamath, and Julia Chuzhoy, editors, *Proccedings of the 52nd Annual ACM SIGACT Symposium on Theory of Computing, STOC 2020, Chicago, IL, USA, June 22-26, 2020*, pages 1125–1138. ACM, 2020. doi:10.1145/3357713.3384277.

25 Anupam Gupta, Amit Kumar, and Debmalya Panigrahi. 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*, pages 504–515. IEEE, 2021. doi:10.1109/FOCS52979.2021.00057.

26 Anupam Gupta, Amit Kumar, and Debmalya Panigrahi. Caching with time windows and delays. *SIAM J. Comput.*, 51(4):975–1017, 2022. doi:10.1137/20m1346286.

27 Kun He, Sizhe Li, Enze Sun, Yuyi Wang, Roger Wattenhofer, and Weihao Zhu. Randomized algorithm for MPMD on two sources. In Jugal Garg, Max Klimm, and Yuqing Kong, editors, *Web and Internet Economics - 19th International Conference, WINE 2023, Shanghai, China, December 4-8, 2023, Proceedings*, volume 14413 of *Lecture Notes in Computer Science*, pages 348–365. Springer, 2023. doi:10.1007/978-3-031-48974-7_20.

28 Sungjin Im, Benjamin Moseley, Chenyang Xu, and Ruilong Zhang. Online dynamic acknowledgement with learned predictions. *CoRR*, abs/2305.18227, 2023. doi:10.48550/arXiv.2305.18227.

29 Anna R. Karlin, Claire Kenyon, and Dana Randall. Dynamic TCP acknowledgment and other stories about e/(e-1). *Algorithmica*, 36(3):209–224, 2003.

30 Predrag Krnetic, Darya Melnyk, Yuyi Wang, and Roger Wattenhofer. The k-server problem with delays on the uniform metric space. In Yixin Cao, Siu-Wing Cheng, and Minming Li, editors, *31st International Symposium on Algorithms and Computation, ISAAC 2020, December 14-18, 2020, Hong Kong, China (Virtual Conference)*, volume 181 of *LIPIcs*, pages 61:1–61:13. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2020. doi:10.4230/LIPIcs.ISAAC.2020.61.

31 Ngoc Mai Le, Seeun William Umboh, and Ningyuan Xie. The power of clairvoyance for multi-level aggregation and set cover with delay. In Nikhil Bansal and Viswanath Nagarajan, editors, *Proceedings of the 2023 ACM-SIAM Symposium on Discrete Algorithms, SODA 2023, Florence, Italy, January 22-25, 2023*, pages 1594–1610. SIAM, 2023. doi:10.1137/1.9781611977554.ch59.

32 Mathieu Mari, Michal Pawlowski, Runtian Ren, and Piotr Sankowski. Online matching with delays and stochastic arrival times. In Noa Agmon, Bo An, Alessandro Ricci, and William Yeoh, editors, *Proceedings of the 2023 International Conference on Autonomous Agents and Multiagent Systems, AAMAS 2023, London, United Kingdom, 29 May 2023 - 2 June 2023*, pages 976–984. ACM, 2023. doi:10.5555/3545946.3598737.

33 Noam Touitou. Nearly-tight lower bounds for set cover and network design with deadlines/delay. In Hee-Kap Ahn and Kunihiko Sadakane, editors, *32nd International Symposium on Algorithms and Computation, ISAAC 2021, December 6-8, 2021, Fukuoka, Japan*, volume 212 of *LIPIcs*, pages 53:1–53:16. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2021. doi:10.4230/LIPIcs.ISAAC.2021.53.

34 Noam Touitou. Frameworks for nonclairvoyant network design with deadlines or delay. In Kousha Etessami, Uriel Feige, and Gabriele Puppis, editors, *50th International Colloquium on Automata, Languages, and Programming, ICALP 2023, July 10-14, 2023, Paderborn, Germany*, volume 261 of *LIPIcs*, pages 105:1–105:20. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2023. doi:10.4230/LIPIcs.ICALP.2023.105.

35 Noam Touitou. Improved and deterministic online service with deadlines or delay. In Barna Saha and Rocco A. Servedio, editors, *Proceedings of the 55th Annual ACM Symposium on Theory of Computing, STOC 2023, Orlando, FL, USA, June 20-23, 2023*, pages 761–774. ACM, 2023. doi:10.1145/3564246.3585107.

A Proof of Lemma 15

To bound the cost of primary services, we first identify and focus on two subsets of services in $\Lambda^{\mathbf{1}}$, namely, $\Lambda^{\mathbf{1,s}}$ and $\Lambda^{\mathbf{1,f}}$.

▶ **Definition 23.** *We define $\Lambda^{1,s} \subseteq \Lambda^{1}$ to be the subset of services in which the server did not move to a new location a'_λ (that is, Line 23 of Algorithm 1 did not run).*

We define $\Lambda^{1,f} \subseteq \Lambda^{1} \setminus \Lambda^{1,s}$ to be the subset of services λ in which $\delta\left(a^*(t_\lambda), a'_\lambda\right) \geq 2^{\ell_\lambda - 7}$; that is, in λ the server moved to a new location that is of distance at least $2^{\ell_\lambda - 7}$ from the optimal solution.

▶ **Proposition 24.** $\sum_{\lambda\in\Lambda^{\mathbf{1}}} 2^{\ell_\lambda} \le O(1)\cdot \mathrm{OPT} + O(1)\cdot\left(\sum_{\lambda\in\Lambda^{\mathbf{1},\mathbf{f}}} 2^{\ell_\lambda} + O(1)\cdot\sum_{\lambda\in\Lambda^{\mathbf{1},\mathbf{s}}} 2^{\ell_\lambda}\right)$.

The proof of Proposition 24 is similar to that of Proposition A.11 in [35]. Intuitively, we again define a potential function proportional to the distance between the algorithm's server and optimal solution's server. For a service $\lambda \in \Lambda^{\mathbf{1}} \setminus (\Lambda^{\mathbf{1},\mathbf{f}} \cup \Lambda^{\mathbf{1},\mathbf{s}})$, the final movement in Line 23 of Algorithm 1 shrinks the distance between the algorithm and the optimal solution such that the resulting decrease in potential is at least 2^{ℓ_λ}.

Proof of Proposition 24. Consider the potential function $\phi(t) := 2^7 \cdot \delta(a(t), a^*(t))$. Note that $\phi(t)$ only takes non-negative values, and is initially 0. The potential function can change upon a movement of the algorithm's server (Line 23 of Algorithm 1) or the optimal solution's server. Note that the total increase in ϕ due to movements in OPT is at most $2^7 \cdot \mathrm{OPT}$; also denote by Δ_λ the change to ϕ due to service $\lambda \in \Lambda^{\mathbf{1}}$ in the algorithm. (Note that in non-primary services, the final location of the server is the same as its initial location, and thus they do not affect the potential function.) Thus, denoting by $\phi(\infty)$ the final value of the potential function, it holds that

$$0 \le \phi(\infty) \le \sum_{\lambda\in\Lambda^{\mathbf{1}}} \Delta_\lambda + 2^7 \cdot \mathrm{OPT}.$$

Adding $\sum_{\lambda\in\Lambda^{\mathbf{1}}} 2^{\ell_\lambda}$ to both sides yields

$$\sum_{\lambda\in\Lambda^{\mathbf{1}}} 2^{\ell_\lambda} \le \sum_{\lambda\in\Lambda^{\mathbf{1}}} (2^{\ell_\lambda} + \Delta_\lambda) + 2^7 \cdot \mathrm{OPT}. \tag{2}$$

First, consider a service $\lambda \in \Lambda^{\mathbf{1}} \setminus (\Lambda^{\mathbf{1},\mathbf{f}} \cup \Lambda^{\mathbf{1},\mathbf{s}})$, and let $t := t_\lambda$. In λ, the server moves from a_λ to a'_λ, and it is guaranteed that $a^*(t) \in B\left(a'_\lambda, 2^{\ell_\lambda - 7}\right)$. Using Proposition 10, it holds $\delta\left(a_\lambda, a'_\lambda\right) \ge 2^{\ell_\lambda-5} - 2^{\ell_\lambda-8}$; thus, $\delta(a_\lambda, a^*(t)) \ge 2^{\ell_\lambda-5} - 2^{\ell_\lambda-7} - 2^{\ell_\lambda-8} \ge 2^{\ell_\lambda-6}$. However, after the movement of the algorithm's server to a'_λ, the distance between the algorithm's server and the optimum's server is at most $2^{\ell_\lambda-7}$; thus, the distance between the servers decreased by at least $2^{\ell_\lambda-7}$, and thus $\Delta_\lambda \le -2^{\ell_\lambda}$. This implies that $2^{\ell_\lambda} + \Delta_\lambda \le 0$ for every $\lambda \in \Lambda^{\mathbf{1}} \setminus \Lambda^{\mathbf{1},\mathbf{f}} \cup \Lambda^{\mathbf{1},\mathbf{s}}$; plugging into Equation (2), we get

$$\sum_{\lambda\in\Lambda^{\mathbf{1}}} 2^{\ell_\lambda} \le 2^7 \cdot \mathrm{OPT} + \sum_{\lambda\in\Lambda^{\mathbf{1},\mathbf{f}}\cup\Lambda^{\mathbf{1},\mathbf{s}}} (2^{\ell_\lambda} + \Delta_\lambda). \tag{3}$$

Again using Proposition 10, for services $\lambda \in \Lambda^{\mathbf{1}}$ where the server moves, it holds that $\delta\left(a_\lambda, a'_\lambda\right) \le 2^{\ell_\lambda-4} + 2^{\ell_\lambda-8} \le 2^{\ell_\lambda-3}$, and thus $\Delta_\lambda \le \cdot 2^{\ell_\lambda+4}$. (When the server does not move, $\Delta_\lambda = 0$.) Thus, it holds that for every $\lambda \in \Lambda^{\mathbf{1}}$, $2^{\ell_\lambda} + \Delta_\lambda \le 17 \cdot 2^{\ell_\lambda}$. Plugging into Equation (3) completes the proof. ◀

▶ **Definition 25.** *For a service $\lambda \in \Lambda^{\mathbf{1},\mathbf{f}} \cup \Lambda^{\mathbf{1},\mathbf{s}}$, define:*

- *R_λ to be the set of pending requests of level at most $\ell_\lambda - 4$ on points in $V^\dagger_\lambda$ at t_λ.*
- *The primary time interval $I_{\mathrm{p}}(\lambda) := (\min_{q\in R_\lambda} r_q, t_\lambda]$.*
- *The primary cylinder $\gamma_{\mathrm{p}}(\lambda) := \left(B\left(a_\lambda, 2^{\ell_\lambda-3}\right), I_{\mathrm{p}}(\lambda)\right)$.*

Further define $V^*_\lambda \subseteq V^\dagger_\lambda$ be the set of locations in $V^\dagger_\lambda$ not visited by the optimal solution during $I_{\mathrm{p}}(\lambda)$. Finally, define the justified residual delay *of λ, which is the amount $D^*_{\mathrm{p},\lambda} := \sum_{v\in V^*_\lambda} y_{\ell_\lambda-4}(v, t_\lambda)$.*

The proofs of the following three propositions appear in Section B.

▶ **Proposition 26.** $\sum_{\lambda\in\Lambda^{\mathbf{1},\mathbf{f}}\cup\Lambda^{\mathbf{1},\mathbf{s}}} D^*_{\mathrm{p},\lambda} \le \mathrm{OPT}$

Proof of Proposition 26. First a service $\lambda \in \Lambda^{\mathbf{1,f}} \cup \Lambda^{\mathbf{1,s}}$, and define $t := t_\lambda$. We first note that the delay cost in $D^*_{\mathrm{p},\lambda}$ is indeed incurred by the optimal solution. Fix a location $v \in V^*_\lambda$; by definition, $y_{\ell_\lambda - 4}(v) = \sum_{\ell \le \ell_\lambda - 4} (g_{v,\ell}(t))^+ \le \sum_{\ell \le \ell_\lambda - 4} \sum_{q \in R'_\ell} d_q(t)$, where $R'_\ell \subseteq R_\lambda$ is the subset of requests on v of level exactly ℓ (this inequality is due to Proposition 9). But, from the definition of $I_{\mathrm{p}}(\lambda)$, the requests of R_λ on v are not served in the optimal solution at t, and thus the optimal solution incurred $\sum_{\ell \le \ell_\lambda - 4} \sum_{q \in R'_\ell} d_q(t)$ delay cost for those requests. Next, note that this delay cost is not charged to the optimal solution for more than one service; this is ensured by the zeroing of the delay counters in Line 17 of Algorithm 1. ◀

▶ **Proposition 27.** *For every $\lambda \in \Lambda^{\mathbf{1,f}} \cup \Lambda^{\mathbf{1,s}}$, it holds that $2^{\ell_\lambda - 8} \le c\big(\mathrm{OPT} \cap \gamma_{\mathrm{p}}(\lambda)\big) + D^*_{\mathrm{p},\lambda}$.*

Proof of Proposition 27. Consider a service $\lambda \in \Lambda^{\mathbf{1,f}} \cup \Lambda^{\mathbf{1,s}}$, and define $t := t_\lambda$, $B := B\big(a_\lambda, 2^{\ell_\lambda - 4}\big)$, $B' := B\big(a_\lambda, 2^{\ell_\lambda - 3}\big)$. Note that B is the ball that contains all requests in R_λ, and that B' is the shape used for the charging cylinder $\gamma_{\mathrm{p}}(\lambda)$. Let a, a^* denote the locations of the algorithm's and optimal solution's servers at t, respectively.

Case 1: $\lambda \in \Lambda^{\mathbf{1,f}}$. Define $a' := a'_\lambda$. Define $B^\star := B\big(a', 2^{\ell_\lambda - 8}\big), B^{\star\star} := B\big(a', 2^{\ell_\lambda - 7}\big)$; note that $B^\star \subseteq B^{\star\star}$. From the choice of a', it holds that $y_{\ell_\lambda - 4}(B^\star) \ge 2^{\ell_\lambda - 5}$. However, $a^* \notin B^{\star\star}$, from the fact that $\lambda \in \Lambda^{\mathbf{1,f}}$. Thus, one of the following holds:

- The optimal solution did not visit $B^\star$ during $I_{\mathrm{p}}(\lambda)$. In this case, it holds that $B^\star \subseteq V^*_\lambda$, and thus $D^*_{\mathrm{p},\lambda} \ge y_{\ell_\lambda - 4}(B^\star) \ge 2^{\ell_\lambda - 5}$, which completes the proof for this case.
- The optimal solution visited $B^\star$ during $I_{\mathrm{p}}(\lambda)$; in this case, the optimal server moved a distance of at least $2^{\ell_\lambda - 7} - 2^{\ell_\lambda - 8} = 2^{\ell_\lambda - 8}$ inside $B^{\star\star} \setminus B^\star$ during $I_{\mathrm{p}}(\lambda)$ (since it was in a^* at the end of $I_{\mathrm{p}}(\lambda)$). Now, note that $B^{\star\star} \subseteq B'$, as Proposition 10 states that $\delta(a', a) \le 2^{\ell_\lambda - 4} + 2^{\ell_\lambda - 8}$; this yields $c\big(\mathrm{OPT} \cap \gamma_{\mathrm{p}}(\lambda)\big) \ge 2^{\ell_\lambda - 7} - 2^{\ell_\lambda - 8} \ge 2^{\ell_\lambda - 8}$, completing the proof.

Case 2: $\lambda \in \Lambda^{\mathbf{1,s}}$. Denote $B^* := B\big(a^*, 2^{\ell_\lambda - 8}\big)$. From the definition of $\Lambda^{\mathbf{1,s}}$, it holds that $y_{\ell_\lambda - 4}(B \setminus B^*) \ge 2^{\ell_\lambda - 5}$. Suppose the optimal solution's server does not visit $B \setminus B^*$ during $I_{\mathrm{p}}(\lambda)$; then, it holds that $B \setminus B^* \subseteq V^*_\lambda$, and thus $D^*_{\mathrm{p},\lambda} \ge 2^{\ell_\lambda - 5}$, completing the proof.

Otherwise, the optimal solution's server visited $B \setminus B^*$ during $I_{\mathrm{p}}(\lambda)$. There are two subcases to consider:

- Suppose $\delta(a^*, a) \le 2^{\ell_\lambda - 3} - 2^{\ell_\lambda - 8}$. In this case, $B^* \subseteq B'$. The optimal solution visited $B \setminus B^*$ during $I_{\mathrm{p}}(\lambda)$, but was at a^* at t; thus, it moved a distance of at least $2^{\ell_\lambda - 8}$ inside B^*, and thus inside B', during $I_{\mathrm{p}}(\lambda)$. This implies that $c\big(\mathrm{OPT} \cap \gamma_{\mathrm{p}}(\lambda)\big) \ge 2^{\ell_\lambda - 8}$, completing the proof.
- Suppose $\delta(a^*, a) > 2^{\ell_\lambda - 3} - 2^{\ell_\lambda - 8}$, and thus the distance of a^* from B is at least $2^{\ell_\lambda - 4} - 2^{\ell_\lambda - 8} \ge 2^{\ell_\lambda - 8}$. Using a similar argument to the previous item, the optimal solution moved a distance of at least $2^{\ell_\lambda - 8}$ inside the ring $B\big(a, 2^{\ell_\lambda - 4} + 2^{\ell_\lambda - 8}\big) \setminus B\big(a, 2^{\ell_\lambda - 4}\big)$ during $I_{\mathrm{p}}(\lambda)$. Observing that this ring is contained in B' yields that $c\big(\mathrm{OPT} \cap \gamma_{\mathrm{p}}(\lambda)\big) \ge 2^{\ell_\lambda - 8}$, completing the proof. ◀

▶ **Proposition 28.** *For every ℓ, the set of cylinders $\big\{\gamma_{\mathrm{p}}(\lambda) \big| \lambda \in \Lambda^{\mathbf{1,f}} \cup \Lambda^{\mathbf{1,s}} \wedge \ell_\lambda = \ell\big\}$ is disjoint.*

Proof of Proposition 28. Suppose for contradiction that there exist $\lambda_1, \lambda_2 \in \Lambda^{\mathbf{1,f}} \cup \Lambda^{\mathbf{1,s}}$ where $\ell_{\lambda_1} = \ell_{\lambda_2} = \ell$ such that $I_{\mathrm{p}}(\lambda_1) \cap I_{\mathrm{p}}(\lambda_2) \neq \emptyset$ and $B\big(a_{\lambda_1}, 2^{\ell - 3}\big) \cap B\big(a_{\lambda_2}, 2^{\ell - 3}\big) \neq \emptyset$. Assume without loss of generality that $t_{\lambda_1} \le t_{\lambda_2}$, and thus $t_{\lambda_1} \in I_{\mathrm{p}}(\lambda_2)$. Then, observe that after λ_1, all pending requests of level at most ℓ in $B\big(a_{\lambda_1}, 2^\ell\big)$ are upgraded to level ℓ. However, in λ_2, the requests in R_{λ_2} are of level at most $\ell - 4$; moreover, the requests in R_{λ_2} are in $B\big(a_{\lambda_2}, 2^{\ell - 4}\big) \subseteq B\big(a_{\lambda_1}, 2^\ell\big)$, where the containment is due to the fact that $\delta\big(a_{\lambda_1}, a_{\lambda_2}\big) \le 2^{\ell - 2}$ (otherwise, we contradict $B\big(a_{\lambda_1}, 2^{\ell - 3}\big) \cap B\big(a_{\lambda_2}, 2^{\ell - 3}\big) \neq \emptyset$). This implies that all requests in R_{λ_2} arrived after t_{λ_1}, yielding a contradiction to $t_{\lambda_1} \in I_{\mathrm{p}}(\lambda_2)$. ◀

Proof of Lemma 15. It holds that

$$\sum_{\lambda\in\Lambda^{\mathbf{1,f}}} 2^{\ell_\lambda} + \sum_{\lambda\in\Lambda^{\mathbf{1,s}}} 2^{\ell_\lambda} \le \sum_{\lambda\in\Lambda^{\mathbf{1,f}}\cup\Lambda^{\mathbf{1,s}}} 2^8\cdot\left(c\left(\mathrm{OPT}\cap\gamma_{\mathrm{p}}(\lambda)\right)+D^*_{\mathrm{p},\lambda}\right)$$
$$\le O(\log k)\cdot\mathrm{OPT}+O(1)\cdot\mathrm{OPT}=O(\log k)\cdot\mathrm{OPT}$$

where the first inequality is due to Proposition 27, and the second inequality is due to Proposition 28 and Proposition 20. ◀

B Omitted Proofs

Proof of Proposition 8. First, suppose a service λ occurs, and consider the point immediately after the loop containing Line 17 of Algorithm 1. At that time, we have $Y_i \le \sum_{v\in B(a_\lambda,2^i)}\sum_{i'\le i}(g_{v,i'})^+ = 0$ for every $i \le \ell_\lambda$. Moreover, for $i > \ell_\lambda$, it holds that $Y_i \le 2^i$ (otherwise, dome i would be critical, and λ would have a higher level as a result).

We now prove the proposition as an invariant over time, noting that at time 0 it trivially holds. Note that the invariant cannot be broken by continuous delay increase; indeed, when Y_i exceeds 2^i, a service is immediately started which zeroes Y_i. Thus, the only possible event that could break the invariant is a movement by the server (Line 23 of Algorithm 1).

Suppose for contradiction that this happens after some (primary) service λ, at a time t^+ immediately after λ, as the server moves from a_λ to a'_λ As before, consider the time during λ immediately after the loop containing Line 17 of Algorithm 1, and denote it by t^-. We also apply the superscripts $-,+$ to Y_i and $g_{v,i}$ to refer to their values at times t^-, t^+ respectively.

Consider any class i. If $i \le \ell_\lambda - 1$, then note that $\delta(a'_\lambda, a_\lambda) \le 2^{\ell_\lambda-4} + 2^{\ell_\lambda-8} < 2^{\ell_\lambda-1}$, due to Proposition 10. Thus, $B(a'_\lambda, 2^i) \subseteq B(a_\lambda, 2^{\ell_\lambda})$. But, for every $v \in B(a_\lambda, 2^{\ell_\lambda})$ and $i' \le \ell_\lambda$, it holds that $g^+_{v,i'} = g^-_{v,i'} \le 0$, and thus $Y^+_i = 0$. Otherwise, $i \ge \ell_\lambda$; in this case, note that $B(a'_\lambda, 2^i) \subseteq B(a_\lambda, 2^{i+1})$. Thus:

$$\begin{aligned}
Y^+_i &\le \sum_{i'\le i} Y^+_{i'} \\
&= \sum_{i'\le i}\sum_{v\in B(a'_\lambda,2^i)} (g_{v,i'})^+ \\
&\le \sum_{i'\le i}\sum_{v\in B(a_\lambda,2^{i+1})} (g_{v,i'})^+ \\
&\le \sum_{i'\le i+1}\sum_{v\in B(a_\lambda,2^{i+1})} (g_{v,i'})^+ \\
&= \sum_{i'\le i+1} Y^-_{i'} \le \sum_{i'\le i+1} 2^{i'} \le 2^{i+2},
\end{aligned}$$

where the first inequality stems from $\{Y_{i'}\}$ being non-negative and the two equalities use the definition of Y_i. The penultimate inequality uses the fact that $Y^-_{i'} \le 2^{i'}$ for every i', as noted in the beginning of the proof. ◀

Proof of Proposition 22. Fixing any ℓ, assume otherwise that there exist two cylinders $\gamma_{\mathrm{c}}(\lambda_1), \gamma_{\mathrm{c}}(\lambda_2) \in \{\gamma_{\mathrm{c}}(\lambda) | \lambda \in \Lambda^{\mathbf{c}} \wedge \ell_\lambda = \ell\}$ that are not disjoint. That is, it holds that $\delta(a_{\lambda_1}, a_{\lambda_2}) < 6\cdot 2^\ell$, and also $I_{\mathrm{c}}(\lambda_1)\cap I_{\mathrm{c}}(\lambda_2) \ne \emptyset$. Let λ'_1, λ'_2 be the level-$(\ell+4)$ secondary services that certify λ_1, λ_2, respectively; without loss of generality, assume λ'_1 occurs before λ'_2.

First, we claim that $\delta\left(a_{\lambda_1}, a_{\lambda'_1}\right) \le 1.5\cdot 2^\ell$. To prove this, consider $q_1 := \sigma_{\lambda'_1}$; it must be that $q_1 \in B(a_{\lambda_1}, 2^{\ell_{\lambda_1}}) \cap B\left(a_{\lambda'_1}, 2^{\ell_{\lambda'_1}-5}\right)$. Since $\ell_{\lambda_1} = \ell$ and $\ell_{\lambda'_1} = \ell+4$, the claim holds.

Next, we claim that λ'_1 occurs before λ_2. Assume otherwise, and consider the request $q_2 := \sigma_{\lambda'_2}$. Since λ'_2 certifies λ_2; moreover, it holds that $q \in B\left(a_{\lambda_2}, 2^{\ell_{\lambda_2}}\right)$. Thus,

$$\begin{aligned}\delta\left(q_2, a_{\lambda'_1}\right) &\leq \delta\left(q_2, a_{\lambda_2}\right) + \delta\left(a_{\lambda_2}, a_{\lambda_1}\right) + \delta\left(a_{\lambda_1}, a_{\lambda'_1}\right) \\ &\leq 2^{\ell} + 6 \cdot 2^{\ell} + 1.5 \cdot 2^{\ell} \leq 8.5 \cdot 2^{\ell} \leq 2^{\ell_{\lambda'_1}}.\end{aligned}$$

Thus, $q_2 \in V_{\lambda'_1}$; moreover, as q was pending at both t_{λ_2} and $t_{\lambda'_2}$, it is pending at $t_{\lambda'_1}$. Due to Line 9 of Algorithm 2, after λ'_1, q_2 is either served or of level $\ell_{\lambda'_1} = \ell + 4$. But this is a contradiction to having level $\ell_{\lambda'_2} - 4 = \ell$ later on, at $t_{\lambda'_2}$; thus, λ'_1 must occur before λ_1.

Thus, λ'_1 occurs between λ_1 and λ_2. Note that λ'_1 occurs after all the tertiary services assigned to λ_1, and thus λ'_1 occurs after $I_c(\lambda_1)$. We claim that λ'_1 also occurs before the release of every request in E_{λ_2}, and thus before $I_c(\lambda_2)$. This claim would yield that $I_c(\lambda_1), I_c(\lambda_2)$ are disjoint, contradicting the assumption that $\gamma_c(\lambda_1), \gamma_c(\lambda_2)$ are disjoint, and would thus conclude the proof of the proposition.

To prove the claim, suppose a request $q \in E_{\lambda_2}$ on some $v \in V_{\lambda_2}$ is released before $t_{\lambda'_1}$. Then, note that $\delta\left(a_{\lambda_2}, a_{\lambda'_1}\right) \leq \delta\left(a_{\lambda_2}, a_{\lambda_1}\right) + \delta\left(a_{\lambda_1}, a_{\lambda'_1}\right) \leq 6 \cdot 2^{\ell} + 1.5 \cdot 2^{\ell} = 7.5 \cdot 2^{\ell}$. Thus, $B\left(a_{\lambda_2}, 2^{\ell_{\lambda_2}}\right) \subseteq B\left(a_{\lambda'_1}, 2^{\ell_{\lambda'_1}}\right)$, and thus $v \in V_{\lambda'_1}$. As q is pending at $t_{\lambda'_1}$, it is of level at least $\ell_{\lambda'_1} = \ell + 4$ after λ'_1; this is in contradiction to $\ell_q \leq \ell$ at λ_2, completing the proof. ◀