Poly-logarithmic Competitiveness for the k-Taxi Problem

Anupam Gupta* Amit Kumar† Debmalya Panigrahi‡

Abstract

The online k-taxi problem generalizes the k-server problem, requiring servers to move between source-sink pairs in an n-point metric space, and the cost is the overhead incurred. In the deterministic setting, the problem has a lower bound on the competitiveness of $\Omega(2^k)$, showing that it is significantly harder than k-server. Randomized algorithms are known with competitiveness $O(2^k \log n)$ (by Coester and Koutsoupias), $O(2^{O(\sqrt{\log \Delta \log k})} \log_\Delta n)$ (by Buchbinder, Coester and Naor), where Δ is the aspect ratio of the n-point metric space), and $O((n \log k)^2 \log n)$ (by Bubeck, Buchbinder, Coester, and Sellke). The best lower bound known is $\Omega(\log^2 k)$ which is inherited from the k-server problem, obtained in a recent breakthrough by Bubeck, Coester, and Rabani, showing a large gap in our understanding of problems that go slightly beyond the metrical task system framework.

An open question left by these works was whether there is a randomized algorithm for the the k-taxi problem with a competitive ratio that is poly-logarithmic in all the parameters. We answer this question in the affirmative in this paper. For our work, we give a covering relaxation for k-taxi on HSTs, which is obtained from the (non-covering) min-cost flow formulation of the problem. The constraints of our LP have compositionality properties that we use to develop a hierarchical primal-dual algorithm defined on the subtrees of the HST.

1 Introduction

In the online k-Taxi problem, we are given an n-point metric space and an online sequence of requests $(\mathbb{s}_q, \mathbb{t}_q)$, each of which comprises a *source* $\mathbb{s}_q$ and a *sink* $\mathbb{t}_q$ in the metric space. There are k taxis available to the algorithm, and to serve request $(\mathbb{s}_q, \mathbb{t}_q)$, the algorithm must move a taxi to source $\mathbb{s}_q$ and then transport it to sink $\mathbb{t}_q$ before the arrival of the next request. The objective of the algorithm is to minimize the total movement cost of the taxis. The famous k-Server problem corresponds to instances of k-Taxi where $\mathbb{s}_q = \mathbb{t}_q$ for every request $(\mathbb{s}_q, \mathbb{t}_q)$. There are two versions of k-Taxi depending on whether the movement cost of taxis *while serving requests* is included in the objective. In the *easy* version of the problem, these costs are included, and there is a simple reduction to k-Server with a constant factor loss in the competitive ratio. The more interesting case is the *hard* version of the problem, which just minimizes the *overhead* involved in the process—i.e., the movements costs of taxis while serving requests is not included. In fact, for *deterministic* algorithms this problem is *exponentially* harder than k-Server: the competitive ratio for k-Taxi is at least $2^k - 1$ [CK19] whereas that for k-Server is between k and $2k - 1$. In the rest of the paper, when we talk of k-Taxi, we mean the hard version of the problem.

The k-Taxi problem was proposed by Karloff and formalized by Fiat, Rabani, and Ravid [FRR90] as a natural generalization of the k-Server problem. Coester and Koutsoupias [CK19] gave a randomized algorithm for the k-Taxi problem with a competitive ratio of $O(2^k \log n)$. Their main tool was a new $(2^k - 1)$-competitive randomized algorithm in *hierarchically separated trees* (HSTs). The result for general metric spaces then follows using standard embedding techniques. They also showed that their competitiveness guarantee for HSTs was true even against adaptive adversaries; this (non-constructively) implies a 4^k-competitive deterministic algorithm for HSTs. Bubeck, Buchbinder, Coester, and Sellke took a different approach of generalizing the k-Taxi problem to a broader problem called *metrical service systems under transformations* (T-MSS), and gave an algorithm for the latter problem [BBCS21]. As a corollary of their T-MSS result, they obtained a competitive ratio of $O((n \log k)^2 \log n)$ for the k-Taxi problem. A third approach was proposed by Buchbinder, Coester, and

*Computer Science Department, Carnegie Mellon University, Pittsburgh, PA, and Google Research. Email: anupamg@cs.cmu.edu. Research partly supported by NSF awards CCF-1955785, CCF-2006953, and CCF-2224718.

†Department of Computer Science and Engineering, IIT Delhi, New Delhi, India. Email: amitk@cse.iitd.ac.in.

‡Department of Computer Science, Duke University, Durham, NC. Email: debmalya@cs.duke.edu. Research partly supported by NSF awards CCF-1750140 (CAREER) and CCF-1955703.

Naor [BCN21]. Using ideas from the classical double coverage algorithm for k-Server, they obtained a k-Taxi algorithm with a competitive ratio of $2^{O(\sqrt{\log \Delta \log k})} \log_{\Delta} n$, where Δ is the aspect ratio of the metric space. While this improves the dependence on (n, k) to subpolynomial, it comes at the cost of a (subpolynomial) dependence on the aspect ratio Δ. To summarize, the previous upper bounds are

$$O(2^k \log n), \;\; O((n \log k)^2 \log n), \;\text{ and }\; 2^{O(\sqrt{\log \Delta \log k})} \log_{\Delta} n.$$

These three results are complemented by a (randomized) lower bound $\Omega(\log^2 k)$, which follows from the recent breakthrough result of Bubeck, Coester, and Rabani for the k-Server problem [BCR23]. This shows a large gap in our understanding of randomized algorithms for k-Taxi and related problems, which do not fall into the standard metrical task system framework, versus k-Server, which does.

These results raise the question: *is there a randomized algorithm (against oblivious adversaries) for the k-Taxi problem with poly-logarithmic competitive ratio, or is the optimal competitiveness qualitatively larger for k-Taxi than for k-Server (as in the deterministic case)?* We make substantial progress on this question and show:

THEOREM 1.1. (RANDOMIZED ALGORITHM FOR k-TAXI) *There is an $O(\log^3 \Delta \cdot \log^2(nk\Delta))$-competitive randomized algorithm for the k-Taxi problem on any n-point metric space with aspect ratio Δ.*

We prove this via Theorem 1.2 below, which gives a fractional solution for k-Taxi on HSTs.

THEOREM 1.2. (FRACTIONAL ALGORITHM FOR k-TAXI ON λ-HSTS) *There is an $O(H^2 \log(nk\Delta))$-competitive fractional algorithm for the k-Taxi problem using $\leq k + 1/2$ servers for any instance on a λ-HST with height H and $\lambda \geq 10H$.*

Any n-point metric space with aspect ratio Δ can be probabilistically embedded into λ-HSTs with height $H = O(\log_{\lambda} \Delta)$ and expected stretch $O(\lambda \log_{\lambda} n)$ [FRT04]; we set $\lambda = O(\log \Delta)$. Then we give a rounding algorithm in Section 6 that is an adaptation of existing rounding algorithms for k-Server from [BBMN15] and [BCL+18]; this shows how to obtain Theorem 1.1 from Theorem 1.2.

An interesting direction for research is to remove the dependence on $\log \Delta$ in our bounds, thereby obtaining the qualitatively best bounds known for k-Server. Nonetheless, we note that no poly-logarithmic guarantees were previously known for k-Taxi in metric spaces with polynomially-bounded aspect ratios; even $O(\text{poly}(k \log n))$ competitiveness was not known for such metrics, let alone for metrics with $\Delta = \exp(k)$, which is the case for some lower-bound examples.

1.1 Our Techniques A central contribution of our work is the formulation of a new linear programming relaxation (LP) for the k-Taxi problem. To the best of our knowledge, all previous algorithms for k-Taxi have taken a combinatorial approach, such as variants of the *double cover* algorithm, or those based on the *work function* algorithm. Unfortunately, even for the special case of k-Server, these combinatorial algorithms have not succeeded in achieving a poly-logarithmic competitive ratio. We make a departure from this line of work and design an LP for the k-Taxi problem, which we describe next.

Offline, the k-Taxi problem can be written as a special case of the min-cost flow problem on a time-expanded metric space. Then, the standard LP for min-cost flow on this time-expanded graph gives a valid LP relaxation for the k-Taxi problem. The difficulty, however, is in obtaining a competitive fractional solution to this LP using an *online algorithm* because (a) the min-cost flow LP is not a covering LP (which is essentially the broadest class of LP we can solve efficiently), and (b) there are up to $O(nT)$ variables in the constraints, and the competitive ratio usually depends on the sparsity of the constraints. Therefore, the challenge in solving the k-Taxi problem, as also special cases such as k-Server and (weighted) paging that were solved previously using LP relaxations, is to obtain the "right" set of constraints in the LP that sufficiently constrain a feasible solution to be able to recover a valid solution to the k-Taxi instance, while also being solvable online. Our LP (obtained by reinterpreting the min-cost flow LP and selecting a subset of constraings from it) addresses the first main challenge. THese constraints are still not sparse, so we need some more care to overcome the second issue—we discuss the ideas behind both steps below.

This approach of using a covering LP relaxation was previously used for the k-Server problem in [GKP21], who also gave a general hierarchical composition framework for solving such LP relaxations in HSTs. However, as mentioned previously, k-Server and k-Taxi have structural differences, and this also manifests in the exponential gap in the deterministic competitive ratios of the two problems—hence we need to introduce new ideas to overcome them. We now recall the main ideas behind the k-Server LP, and then discuss the differences and the the derivation of the k-Taxi LP from first principles.

LP **relaxation for k-Server [GKP21].** This relaxation has variables of the form $x(v,t)$, where v is a node of the HST and t is a time, which denote the number of servers that traverse the edge joining v to its parent $p(v)$ at time t. Now consider a set A of $(k+1)$ leaves along with one request at each of these leaf nodes. At least two of these requests must be served by the same server. Say the corresponding requests are at times q_1, q_2 at leaves $\ell_1, \ell_2 \in A$, with $q_1 < q_2$. So the same server is present at ℓ_1 at time q_1 and at ℓ_2 at time q_2 (note that it could visit some arbitrary subset of leaves in the intervening period). Let P denote the path in the tree from ℓ_1 to the least common ancestor v of ℓ_1 and ℓ_2, with the edges (from bottom to top) being $e_1, \ldots, e_r$. Now, if we partition the time interval $(q_1, q_2]$ into r sub-intervals $I_1, \ldots, I_r$, then it must be the case that the server traverses one of the edges e_i during the corresponding interval I_i. This observation can be codified as a covering constraint involving variables $x(v,t)$, for v being a vertex in the minimal sub-tree containing A, and t being a suitable time depending on the request arrival times at A.

Our LP **relaxation for k-Taxi.** Our approach for k-Taxi also involves writing a suitable LP relaxation. However, unlike k-Server, where all server movements are paid for in the LP objective, there are two types of server movement in k-Taxi: *with* or *without* passengers. Only movements of the second type form part of the LP objective and thus the variables $x(v,t)$ only correspond to this type of server movement. More precisely, using the notation above, recall that in the case of k-Server, we know that the server would traverse each edge of path P at some time during $(q_1, q_2]$, and so we could write a suitable covering relaxation. Now, in k-Taxi, even though the server will traverse this path, it might do so *while carrying passengers* from the source to the sink of a request (we call this *short-cutting*). This type of movement is not captured by the $x(v,t)$ variables, and hence, a covering constraint written on these variables for this path would be invalid. Indeed, suppose there is a server serving two requests q_1, q_2, where the first request has its sink at leaf ℓ_1 and the second one has its source at ℓ_2. Further assume that there is another request q between q_1, q_2 such that the source of q is close to ℓ_1 and the sink of q is close to ℓ_2. Then the server can go from ℓ_1 to ℓ_2 at a low cost by short-cutting through the request q.

Let us fix a vertex v and consider the subtree $\mathbb{T}_v$ under it in the HST, and fix any time t. Now, the requests after time t can be partitioned into four groups depending on whether the source and sink location is in the subtree or outside it. Of these, requests where both the source and sink vertex are in the subtree, or both are outside, do not impose any constraints on server movement from v to its parent $p(v)$ on the $(v, p(v))$ edge. We ignore these requests in the rest of the discussion.

We are left with two types of requests: those with only the sink node in $\mathbb{T}_v$ and only the source node in $\mathbb{T}_v$. Let us call these requests of type A and type B respectively. If a single server serves two requests of the type A, any feasible solution would need to move the server on the $(v, p(v))$ between serving these two requests. More generally, if there are k servers and $k+r$ requests of type A, a feasible solution would incur r movements on the $(v, p(v))$ edge. But, if these requests of type A are punctuated by requests of type B, then the server gets a "free ride" out of subtree $\mathbb{T}_v$ when it is transported to the sink of this request (outside the subtree) at zero cost. Hence, in general, a valid constraint on the $(v, p(v))$ edge is given by the *difference* between requests of type A and type B. To encode constraints of this form, we use a system of timestamps on the nodes of the HST. Based on these timestamps and the difference in the number of requests of the two types described above in relation to these timestamps, we derive a valid set of constraints on the server movements out of subtrees in the HST (i.e., on the edge from the root of the subtree to its parent) captured by the variables $x(v,t)$.

A two-step process for serving requests. Next, we describe how we simulate the movement of one unit of server mass from the source to the sink of a request. We do this in two steps. In the first step, we increase the server mass at the sink node by one unit and correspondingly decrease the server mass as the source node by one unit. This might lead to a negative server mass at the source node. In the second step, we restore the non-negativity of server mass at the source node by moving server mass from other nodes to the source node.[1] Server movements in the second step are captured by the $x(v,t)$ variables and define the constraints and objective of our LP. This two-step process creates an additional complication: we need to ensure that the additional server

[1]For technical reasons, we do not insist on strict non-negativity of server mass at a node, and allow the server mass to the *slightly* negative, but this is not crucial for this high level description.

mass situated at the sink at the end of the first step is not transferred back to the source in the second step because this would not represent a valid servicing of the request. To this end, we create a dummy *holding node* at the sink from which server is not allowed to move out in the second step. At the end of the second step, we merge the holding node at the sink with the actual sink node since the server mass at the sink is now allowed to move to any other location to service a future request.

Truncation and composition of constraints. One key feature of the algorithmic framework in [GKP21] is the notion of *local* LPs: for each internal node v, we obtain a new local LP by *truncating* each constraint of the actual (i.e., global) LP relaxation. We explain this concept informally here (see... for more technical details). The covering LP relaxation in [GKP21] has a constraint for each subset $A = (v_1, \ldots, v_h)$ of leaves of the underlying tree and each sequence $(t_1, \ldots, t_h)$ of request times at these leaves. The covering constraint is of the form:

$$\sum_{v \in T^A} \sum_{t \in S(v)} x_{v,t} \geq k - |A| + 1. \tag{1.1}$$

Here T^A is the sub-tree induced by leaves A and the root, and $S(v)$ is some set of timesteps associated with each vertex v. (This subset $S(v)$ just depends on the coordinates of the vectors $(v_1, \ldots, v_h), (t_1, \ldots, t_h)$ corresponding to vertices of A below v). Now a truncated constraint at a vertex w is obtained by considering the variables on the LHS that correspond to the sub-tree below w. More precisely, the truncated constraint at a vertex w is given by:

$$\sum_{v \in T^A(w)} \sum_{t \in S(v)} x_{w,t} \geq k_w - |A_w| + \varepsilon_w,$$

where $T^A(w)$ is the sub-tree of T^A below w, k_w is the number of servers below w at a certain time (which is the maximum of all the t_i below w), $|A_w|$ is the set of vertices of A below w, and ε_w is a small parameter depending on number of nodes below w. The key observation is that given the actual constraint (1.1), the corresponding truncated constraint can be written down in this mechanical manner. Note that these are not valid constraints (since they are defined based on the algorithm's behavior), but they help us keep track of the server mass movement in the tree.

This gives the following observation: given the truncated constraint at a node w, we can write down the corresponding truncated constraint at any child u of w. Conversely, *composition* asks the following: given the truncated constraints at each child of a node w, can we write the corresponding truncated constraint at w? Suppose the children of w are $u_1, \ldots, u_h$, then the truncated constraint at u_i depends on the part of the vectors $(v_1, \ldots, v_h), (t_1, \ldots, t_h)$ which corresponding to nodes in A below u_i. So intuitively it may seem that given these partial vectors, we can concatenate them to infer the truncated constraint at w. However, in order to carry out this plan, some additional conditions are needed. [GKP21] achieved this by re-writing the constraints in a different form. We show that a similar composition rule holds here; in fact, our composition simplifies and extends the presentation of [GKP21]: each truncated constraint a node u is simply given by a sequence of times at each node in the subtree below u. Given a subset X of children of v and a truncated constraint for each of them, the composed constraint at v is obtained by just taking the concatenation of the corresponding time vectors at each node in X (and using a default value for other nodes in the sub-tree below v). This composition illustrates the main idea behind the local LPs: the movement within a subtree rooted at node v is charged to the local LP at v. Moreover, in our analysis we relate the duals for each local LP to those of its children, thereby transferring this account scheme up the tree, until it reaches the root. At the root, the local LP constraints are indeed valid, and hence the relationship between the primal and dual movements that we maintain inductively up the tree completes our analysis.

Although these truncated constraints (at a node v) are simple to state (and the composition is even simpler than before), analysing them requires rephrasing them in terms of server movement across the sub-tree below v. A major subtlety, which was not present in [GKP21], is that server mass can enter the subtree below v not only through the parent edge of v, but also through the short-cutting process (where one unit of server mass directly goes from a source request to the corresponding sink request). This requires additional care in the analysis of the algorithm.

Controlling the size of the LP. In standard online algorithms for covering problems, the competitive ratio depends on the maximum number of variables appearing in any constraint. For our LP relaxation, a constraint could have T variables, where T is the length of time horizon, and so the competitive ratio could potentially depend on $\log T$. [GKP21] get around this issue by showing that each constraint only needs to involve requests which arrived in the recent past (polynomial in the size of the HST). This argument critically uses the fact that servers can enter the subtree below a vertex v only through the edge joining v to its parent. This fact is no longer true for k-Taxi because of short-cutting. Instead, we show that when a covering constraint has a lot of variables, the optimal value must have also increased significantly. In such a scenario, we *restart* the online algorithm from scratch (see §5.4 for details). This ensures that the size of each covering constraint in our LP remains polynomially bounded in the size of the HST.

1.2 Roadmap In §2, we describe the covering LP relaxation for k-Taxi, and explain how it derives from a flow-based LP formulation. In §3 we define the notion of "truncated" constraints used to define local LPs at the internal nodes of the HST, and show how constraints for the children's local LPs can be composed to get constraints for the parent LP. We then give the algorithm and analysis for the k-Taxi problem in §4 and §5 respectively. The rounding algorithm is given in §6.

1.3 Related Work The k-Taxi problem was originally proposed by Karloff, who gave a competitive algorithm for $k = 2$ (see [FRR90]). Fiat, Rabani, and Ravid [FRR90] introduced the k-Taxi problem for general k; they also formally distinguished between the easy and hard versions, giving a competitive algorithm for the easy version by adapting their k-Server algorithm and stating the hard version as an open problem. Coester and Koutsoupias [CK19] established the $2^k - 1$ deterministic lower bound for the hard version and also gave an $O(2^k \log n)$-competitive randomized algorithm for the problem; all subsequent works are cited in the introduction. A *stochastic* version of the easy k-Taxi problem was studied as the Uber problem in [DEH+17]. We note that previous results for (hard) k-Taxi rely on combinatorial techniques such as double cover and work functions, based on which a better-than-k competitive algorithm is not known even for the special case of k-Server. We take a different approach in this paper by designing an LP relaxation for the k-Taxi problem; this change of approach was crucial in breaking through the barrier of k for the k-Server problem as well.

Early work on the closely related k-Server problem focused on deterministic algorithms [FRR94, KP95], and on combinatorial randomized algorithms [Gro91, BG00]. This problem was also studied for special metric spaces, such as lines, (weighted) stars, trees: e.g., [CKPV91, CL91, FKL+91, MS91, ACN00, BBN12a, Sei01, CMP08, CL06, BBN12b, BBN10]. Works obtaining a poly-logarithmic competitive ratio are more recent, starting with [BBMN15], and more recently, by [BCL+18].These works, and indeed most randomized algorithms for k-Server, start with algorithms for instances defined on HSTs, and then use metric embedding techniques to extend the result to general (finite) metric spaces. A new and remarkable LP relaxation was introduced by [BCL+18], who then use a mirror descent strategy with a multi-level entropy regularizer to obtain the online dynamics. [BGMN19] gives an alternate projection-based perspective on [BCL+18]. Unfortunately, these LP relaxations do not appear to have natural extensions to the k-Taxi problem.

Our work builds on the approach of [GKP21], who gave a novel covering LP relaxation for the k-Server problem on HSTs and devised an online fractional set cover-style algorithm for both k-Server. The competitive ratio of their fractional solution was $O(\log^2(n\Delta))$, where Δ is the aspect ratio of the metric space. A major contribution of this work is the development of a new covering LP relaxation, which allows us to extend the framework from [GKP21] for k-Taxi; moreover, we need new ideas to go from $\log T$ to $\log \Delta k$, over and above those from [GKP21].

2 A Covering LP Relaxation for k-Taxi

The k-Taxi problem can be formulated as a min-cost flow instance, so it is not surprising that we can write a LP relaxation for it. However, we want our relaxation to be a covering LP, since covering LPs are essentially the most general class we know how to solve online. Moreover, as we mentioned in the introduction, the k-Taxi problem has an unusual cost structure (where some of the moves are free and others incur costs); this poses a challenge. Our LP relaxation identifies a set of constraints implied by the min-cost flow formulation (but are not equivalent to it), which nevertheless suffice to give us a good online algorithm.

2.1 Notation and Preliminaries We consider the k-Taxi problem on a hierarchically separated tree (HST) $\mathbb{T}$ having height H, rooted at node $\mathbb{r}$ and having n leaves. Define the *level* of a node as its height, with the leaves at level 0, and the root at level H. For a non-root node v, the length of the edge $(v, p(v))$ going to its parent $p(v)$ is $\lambda^{\text{level}(v)}$, and is denoted by c_v. This means that leaf edges have length 1, whereas edges from the root to its children have length λ^{H-1}. We assume that $\lambda \geq 10H$; we set $\lambda, H = O(\log \Delta)$. For a vertex v, let χ_v be its children, $\mathbb{T}_v$ be the subtree rooted at v, and L_v be the leaves in this subtree. Let $n_v := |\mathbb{T}_v|$.

Request Times and Timesteps. Let the request sequence be $\mathfrak{R} := r_1, r_2, \ldots$, where each request $r_q \in \mathfrak{R}$ is a source-sink tuple $(\mathbb{s}_q, \mathbb{t}_q)$ for some leaves $\mathbb{s}_q, \mathbb{t}_q$ and distinct *request time* $q \in \mathbb{Z}_+$. A solution must ensure that a server/taxi visits $\mathbb{s}_q$ at time q, and then this server gets transferred to $\mathbb{t}_q$ at no additional cost. Between any two *request times* q and $q+1$, we define a collection of *timesteps* (each denoted by τ or t)—these timesteps take on values $\{q + \eta \cdot j\}$ for some small value $\eta \geq 0$ and $j = 1, 2, \ldots$; each request arrival time $q \in \mathbb{Z}$ is also a timestep. We use $\mathfrak{T}$ to denote the set of all timesteps. Given a timestep τ, let $\lfloor \tau \rfloor$ refer to the request time q such that $\tau \in [q, q+1)$. The notion of fractional timesteps will not be crucial for the flow formulation, but will make the description of our subsequent online algorithm cleaner. Indeed, our online algorithm will transfer fractional server mass to $\mathbb{s}_q$ in tiny amounts, and we can keep track of this process by designating a unique timestep with each such transfer.

2.2 Min-cost Flow formulation for k-Taxi We start by describe this formulation of the off-line problem using min-cost flows. Consider an instance of the k-Taxi problem consisting of a sequence of N request times. Recall that request r_q is given by a pair $(\mathbb{s}_q, \mathbb{t}_q)$ of points in $\mathbb{T}$, and a time t. The underlying graph in the flow formulation is a time-indexed graph, where we have a copy of the vertex set V for each timestep $\mathfrak{T}$. (As mentioned above, we could give the flow formulation just by having a copy V_q of the vertex set V for each request time $q \in \mathbb{Z}$, but the fractional timesteps in $\mathfrak{T}$ will be useful subsequently.) We assume (w.l.o.g.) that for each request q, a server arrives (or is already present) at $\mathbb{s}_q$ at timestep q and is present at $\mathbb{t}_q$ at timestep $q+1$.

The vertex set $V(G)$ of the time-expanded graph G has one copy of V for each timestep τ, plus special source and sink vertices s_G and t_G respectively, i.e.,

$$V(G) := \{(v, t) \mid v \in V(\mathbb{T}), t \in \mathfrak{T}\} \cup \{s_G, t_G\}.$$

The directed edges $E(G)$ in G are of three types (there are no edge-capacities):

(i) cost-0 edges $\{(s_G, (v, 1)), ((v, N), t_G) \mid v \in V(\mathbb{T})\}$ connecting the source and sink to the first and last copies of each node,

(ii) cost-0 edges $\{((v, t), (v, t^+)) \mid v \in V(\mathbb{T}), t \in \mathfrak{T}\}$ between consecutive copies of the same vertex (here t^+ denotes the timestep after t), and

(iii) edges $\{((v, t), (p(v), t)) \mid v \in V(\mathbb{T}), t \in \mathfrak{T}\}$ of cost c_v between each node and its parent, and $\{((v, t), (u, t)) \mid v \in V(\mathbb{T}), u \in \chi_v, t \in \mathfrak{T}\}$ of cost zero between a node and its children. (This captures that moving servers up the tree incurs cost, but moving down the tree can be done free of charge.)

For each vertex $w \in V(G)$, we define its supply as follows. The source s_G has k units of supply, and sink t_G has k units of demand (or equivalently, a supply of $-k$). For each request q, the node $(\mathbb{s}_q, q)$ has one unit of demand (and hence -1 supply), and node $(\mathbb{t}_q, q+1)$ has one unit of supply. All other vertices in $V(G)$ have zero supply.

The integrality of the min-cost flow polytope implies that an optimal solution to this transportation problem captures the optimal k-Taxi solution. Moreover, the max-flow min-cut theorem says that if $\{x(e)\}_{e \in E(G)}$ is a solution to this transportation problem, then for all subsets $S \subseteq V(G)$ we have

$$x(\partial^+(S)) \geq \mathsf{supply}(S).$$

Thus we can frame the k-Taxi problem as the following covering problem:

$$\text{(LP:flow)} \qquad \min \sum_{v \in V(\mathbb{T}) \setminus \{\mathbb{r}\}} \sum_{\tau \in \mathfrak{T}} c_v \cdot x((v, \tau), (p(v), \tau))$$

$$\text{(2.2)} \qquad \text{s.t.} \quad x(\partial^+(S)) \geq \mathsf{supply}(S) \quad \forall S \subseteq V(G)$$

$$x(e) \geq 0 \quad \forall e \in E(G).$$

Moreover, we can use techniques from online covering LP to maintain a fractional solution to the above LP [BN09, GN14]. However, two issues arise:

(a) At some timestep τ the online LP solution may raise the variable $x((v, \tau'), (p(v), \tau'))$ corresponding to some timestep $\tau' \ll \tau$, and it is not clear how the online algorithm should process this movement of mass "occurring in the past".

(b) Moreover, the constraints in (2.3) may have many variables, as many as the length N of the input: this can cause the competitive ratio of the online LP solvers to depend on $\log N$, which could be much larger than $\log(n\Delta)$: we want to avoid any dependence on the time horizon N.

Our next contribution is to carefully select a subset of these covering constraints that avoids the problems above, which are nonetheless rich enough to give an online fractional solution (and by rounding, an integral solution) to the k-Taxi instance. We describe this covering LP relaxation now.

2.3 The Covering LP Relaxation The covering LP relaxation ($\mathfrak{M}$) we use for k-Taxi is obtained from (LP:flow) by selecting a suitable subset of the constraints (2.2) using the notion of *monotone vectors* of timesteps.

Definition 1. (Monotone Vector $\boldsymbol{\tau}$) *Let $\boldsymbol{\tau} \in \mathfrak{T}^{|V|}$ be a vector of timesteps, with one coordinate τ_u for each node u in the tree $\mathbb{T}$. The vector $\boldsymbol{\tau}$ is* monotone *if $\tau_u \le \tau_{p(u)}$ for all non-root nodes u.*

Given such a monotone vector $\boldsymbol{\tau}$, consider the following subset $S_{\boldsymbol{\tau}}$ of $V(G)$: we add t_G to $S_{\boldsymbol{\tau}}$. For each $u \in V(G)$, we add vertices $(u,\tau), \tau > \tau_u$, to $S_{\boldsymbol{\tau}}$. Thus,

$$S_{\boldsymbol{\tau}} := \{t_G\} \cup \bigcup_{u \in V(G)} \{(u,\tau) : \tau > \tau_u\}.$$

Our LP relaxation $\mathfrak{M}$ is obtained by adding the constraint (2.2) corresponding to $S_{\boldsymbol{\tau}}$ for each monotone vector $\boldsymbol{\tau}$. In the rest of this section, we interpret the the terms on both sides of constraint (2.2) for such a set $S_{\boldsymbol{\tau}}$. To do this, define two subsets of requests based on any monotone vector $\boldsymbol{\tau}$ as follows:

- $P(\boldsymbol{\tau})$ is the set of requests q for which $\lfloor \tau_{\mathbb{t}_q} \rfloor \le q$, and
- $N(\boldsymbol{\tau})$ is the set of requests q for which $\lfloor \tau_{\mathbb{s}_q} \rfloor < q$.

Observe the differences in the two definitions: the latter is defined using $\tau_{\mathbb{s}_q}$ instead of $\tau_{\mathbb{t}_q}$, and moreover it uses a strict inequality. These differences will be crucial for our LP. To get some intuition for the strict inequality, consider the special case of the k-Server problem where $\mathbb{s}_q = \mathbb{t}_q$ for all q: here $|P(\boldsymbol{\tau})| - |N(\boldsymbol{\tau})|$ equals the number of locations u which receive a request exactly at time τ_u. If we choose τ_u at each leaf u to equal a request time at that leaf, this makes $|P(\boldsymbol{\tau})| - |N(\boldsymbol{\tau})| = n$. Moreover, we need the asymmetry between $\mathbb{s}_q$ and $\mathbb{t}_q$ in the setting in k-Taxi.

Given these definitions, let us reinterpret constraint (2.2): we first consider the RHS.

Claim 2.1. *For a monotone vector $\boldsymbol{\tau}$, we have* $\mathsf{supply}(S_{\boldsymbol{\tau}}) = |P(\boldsymbol{\tau})| - |N(\boldsymbol{\tau})| - k$.

Proof. The supply of t_G is $-k$. Now consider a request q. The vertex $(\mathbb{t}_q, q+1) \in S_{\boldsymbol{\tau}}$ iff $\lfloor \tau_{\mathbb{t}_q} \rfloor \le q$, which is same as the condition that $q \in P(\boldsymbol{\tau})$. Similarly, the vertex $(\mathbb{s}_q, q) \in S_{\boldsymbol{\tau}}$ iff $q \in N(\boldsymbol{\tau})$. Since each of the vertices $(\mathbb{t}_q, q+1)$ has $+1$ supply, and each of the vertices $(\mathbb{s}_q, q)$ has -1 supply, the desired result follows. ∎

Next we look at the LHS of constraint (2.2) for $S_{\boldsymbol{\tau}}$. For brevity, we use $x(v,\tau)$ to denote $x((v,\tau),(p(v),\tau))$; given an interval I of timesteps, we use $x(v,I)$ to denote $\sum_{\tau \in I} x(v,\tau))$.

Claim 2.2. *Let $\boldsymbol{\tau}$ be a monotone vector. Then,*

$$x(\partial^+(S_{\boldsymbol{\tau}})) = \sum_{v \in V(\mathbb{T}) \setminus \{\mathbb{r}\}} \sum_{\tau \in (\tau_v, \tau_{p(v)}]} x(v,\tau) = \sum_{v \in V(\mathbb{T}) \setminus \{\mathbb{r}\}} x(v, (\tau_v, \tau_{p(v)}]).$$

Proof. Suppose an edge $e = ((v,\tau),(v',\tau')) \in \partial^+(S_{\boldsymbol{\tau}})$. We know that $\tau' > \tau_{p(v)} \ge \tau_v$. Three cases arise:

- $(v',\tau') = (v,\tau^+)$: Since $\tau^+ > \tau > \tau_v$, $(v,\tau^+) \in S_{\boldsymbol{\tau}}$ as well. Therefore, $e \notin \partial^+(S_{\boldsymbol{\tau}})$. Thus, this case cannot occur.
- v' is a child of v: In this case, $\tau' = \tau$. But then, $\tau = \tau' > \tau_{p(v)} \ge \tau_v$. But then, $(v',\tau') \in S_{\boldsymbol{\tau}}$ as well, a contradiction.
- $v' = p(v), \tau' = \tau$: For this to happen, $(p(v),\tau)$ should not lie in $S_{\boldsymbol{\tau}}$, i.e., $\tau \le \tau_{p(v)}$.

The above observations show that the edge $e = ((v,\tau),(v',\tau')) \in \partial^+(S_{\boldsymbol{\tau}})$ iff $\tau = \tau'$ and $\tau \in (\tau_v, \tau_{p(v)}]$. This proves the desired result. ∎

Using Claims 2.1 and 2.2 gives our new relaxation for k-Taxi, which we use in the rest of the paper.

($\mathfrak{M}$) $$\min \sum_{v \in V(\mathbb{T}) \setminus \{\mathbb{r}\}} \sum_{\tau \in \mathfrak{T}} c_v \cdot x(v,\tau)$$

$$(2.3) \qquad \text{s.t.} \sum_{v \in V(\mathbb{T}) \setminus \{\mathbb{r}\}} x(v, (\tau_v, \tau_{p(v)})) \geq |P(\boldsymbol{\tau})| - |N(\boldsymbol{\tau})| - k \qquad \text{for all monotone vectors } \boldsymbol{\tau}$$

$$x(v, \tau) \geq 0 \qquad \forall v \in V(\mathbb{T}) \setminus \{\mathbb{r}\}, \tau \in \mathfrak{T}.$$

3 The Local LPs: Truncation and Composition

In addition to the main LP relaxation $\mathfrak{M}$, we can write a version of the LP for each subtree that is rooted at some internal vertex v of $\mathbb{T}$. (We call this the *local* LP $\mathfrak{L}^v$ for vertex v; the local LP for the root $\mathbb{r}$ contains a subset of constraints from the global LP $\mathfrak{M}$.) These local LPs look fairly similar to $\mathfrak{M}$, with some crucial differences: the obvious one is that the monotone vectors are restricted to vertices in the subtree $\mathbb{T}_v$.

A very important—and unusual—feature is that each non-root local LP is itself defined based on *our algorithm's* state, and not just the problem instance. Consequently, the constraints of these local LPs are not necessarily valid for the original k-Taxi instance (with the exception of $\mathfrak{L}^{\mathbb{r}}$, which contains constraints from $\mathfrak{M}$). On the other hand, we get an all-important *compositionality property*: for each node, the constraints of the local LPs for v's children can be composed together to give constraints of the v's local LP. Moreover, the dual solutions for the children LPs can be combined to give valid duals for the parent's LP. Hence if we relate (a) the movement cost incurred within each subtree $\mathbb{T}_v$ to its local LP $\mathfrak{L}^v$, and (b) the cost of each local LP to its children LPs, we can inductively relate the total movement cost to the cost of the root LP, and hence to the global LP $\mathfrak{M}$ and the optimal solution for the k-Taxi instance.

3.1 Some Terminology Before we define the local LPs, we introduce some notation and conventions. We classify each leaf node as either a *source* or a *sink* node: a source leaf node can only act as $\mathbb{s}_q$ for some request q, and similarly a sink leaf node can only act as $\mathbb{t}_q$. Moreover, we imagine that each sink node $\mathbb{t}_q$ creates another copy $\mathbb{t}'_q$ co-located with it; this copy is called the *holding node*. When we see a request $(\mathbb{s}_q, \mathbb{t}_q)$ at some time q:

1. We first transfer one unit of mass from $\mathbb{s}_q$ to the holding node $\mathbb{t}'_q$ at zero cost. This typically results in *negative server mass* at $\mathbb{s}_q$.

2. Next, we transfer server mass from sink nodes (except from the holding node $\mathbb{t}'_q$) to $\mathbb{s}_q$ until $\mathbb{s}_q$ has server mass close to zero (albeit still negative). This clarifies the reason for the holding node: any server present at $\mathbb{t}_q$ prior to this request may be sent to $\mathbb{s}_q$ as needed, but the unit of server mass we just sent to $\mathbb{t}'_q$ from $\mathbb{s}_q$ should not be routed back to the source $\mathbb{s}_q$.

3. Finally, we move the unit server mass from the holding node $\mathbb{t}'_q$ to the sink $\mathbb{t}_q$ (again at zero cost, since they are co-located), and imagine the holding node $\mathbb{t}'_q$ as having disappeared.

The actual (fractional) movement of server mass to $\mathbb{s}_q$ in step 2 will be done by our (fractional) algorithm $\mathcal{A}$ by transferring a small amount of server mass in each timestep, as we explain in the rest of this section.

We choose two non-negative parameters, δ and γ. The parameter δ helps define bounds on the amount of (fractional) servers at any leaf, whereas γ is used to define the granularity of moving server mass. We ensure $\delta \gg \gamma$: specifically, we set $\delta = \frac{1}{10n^3}$ and $\gamma = \frac{1}{n^4}$.

Definition 2. (Active and Saturated Leaves) *Given an algorithm $\mathcal{A}$, a sink leaf ℓ is* active *if it is not a holding node and has at least δ amount of server (and* inactive *otherwise). A source leaf is* saturated *if it has server mass at least $-4\delta n$, and* unsaturated *otherwise.*

We maintain the following invariant:

Invariant (I1). *The server mass at each source leaf lies in the interval $(-1 - 4\delta n, -2\delta n]$, whereas the server mass at each sink leaf is at least $\delta/2$. Moreover, after servicing a request, the server mass at a source leaf lies in the range $(-4\delta n, -2\delta n]$.*

3.2 Truncated Constraints and their Composition As a first step towards writing down the local LPs $\mathfrak{L}^v$, we define constraints based on the state of our k-Taxi algorithm $\mathcal{A}$ and its actions. Let $k_{v,t}$ be the server mass that $\mathcal{A}$ has in v's subtree $\mathbb{T}_v$ at the end of timestep t (when v is a leaf, this just denotes the amount of server mass at v at the end of timestep t). The constraints of $\mathfrak{L}^v$ are defined using *truncation* of the constraints $\varphi_{\boldsymbol{\tau}}$. Recall that $\mathbb{T}_v$ is the sub-tree of $\mathbb{T}$ under node v.

Definition 3. (Truncated Constraints) *Consider a node v and a monotone timestep vector $\boldsymbol{\tau} := (\tau_u)_{u\in\mathbb{T}_v}$. (I.e., it satisfies that $\tau_u \le \tau_{p(u)}$ for all $u \in \mathbb{T}_v \setminus \{v\}$.) Suppose there exists a source request $\mathbb{s}_q$ below v with request time $\lfloor\tau_v\rfloor$. Define $P_v(\boldsymbol{\tau})$ as the multi-set of sink leaf nodes $\mathbb{t}_q \in \mathbb{T}_v$ with $\lfloor\tau_{\mathbb{t}_q}\rfloor \le q \le \tau_v$, and $N_v(\boldsymbol{\tau})$ as the multi-set of source leaf nodes $\mathbb{s}_q \in \mathbb{T}_v$ satisfying $\lfloor\tau_{\mathbb{s}_q}\rfloor < q \le \tau_v$. Note that we only consider requests that occur by timestep τ_v. The* truncated constraint *$\varphi_{\boldsymbol{\tau},v}$ is defined as:*

$$\sum_{u\in V(\mathbb{T}_v)\setminus\{v\}} y^v(u,(\tau_u,\tau_{p(u)}]) \ge |P_v(\boldsymbol{\tau})| - |N_v(\boldsymbol{\tau})| - k_{v,\tau_v} - 2\delta(n-n_v); \tag{3.4}$$

recall that k_{v,τ_v} is the amount of server mass in $\mathbb{T}_v$ at the end of *timestep τ_v. We say that the truncated constraint $\varphi_{\boldsymbol{\tau},v}$* ends at *$\tau_v$.*

We emphasize: these constraints do not use the global variables x, but instead *local variables* y^v that are "private" to node v. Moreover, truncated constraints are *not necessarily implied* by the LP relaxation $\mathfrak{M}$ even when we replace y^v by x, since they are defined based on $\mathcal{A}$'s choices, and a generic algorithm is not constrained to maintain k_{v,τ_v} servers in subtree $\mathbb{T}_v$ after timestep τ_v. On the other hand, when $v = \mathbb{r}$ we have $k_{\mathbb{r},\tau_{\mathbb{r}}} = k$ and the last term is 0, and we get back the constraints from $\mathfrak{M}$.

Composing Constraints. The local constraints at the children of any node v can be composed to give constraints that are implied by the constraints at v. Indeed, consider a non-empty subset X of v's children. For $u \in X$, let $\varphi_{\boldsymbol{\tau}(u),u}$ be a constraint in $\mathfrak{L}^u$ ending at $\tau_u := \boldsymbol{\tau}(u)_u$. Let $\overline{X}$ denote the children of v that are not in X. Then we can define the vector $\boldsymbol{\tau} : \mathbb{T}_v \to \mathfrak{T}$ by extending the vectors $\boldsymbol{\tau}(u)$ to all nodes in $\cup_{u\in X}\mathbb{T}_u$ by setting $\tau_v = \max_{u\in X}\tau_u$ and $\tau_w = \tau_v$ for all $w \in \cup_{u\in\overline{X}}\mathbb{T}_u$. This yields the local constraint $\varphi_{\boldsymbol{\tau},v}$ at $\mathfrak{L}^v$ ending at τ_v. We say that $\varphi_{\boldsymbol{\tau},v}$ has been obtained by *composition* of the constraints $\varphi_{\boldsymbol{\tau}(u),u}, u \in X$.

3.3 Timesteps, Constraint Sets, and the Local LPs A first attempt to define the local LP $\mathfrak{L}^v$ would be to minimize $\sum_{v\in V(\mathbb{T}_v)\setminus\{v\}}\sum_{\tau\in\mathfrak{T}} c_v \cdot y^v(v,\tau)$ subject to all constraints of the form $\varphi_{\boldsymbol{\tau},v}$ for all monotone vectors $\boldsymbol{\tau}$ defined over the subtree $\mathbb{T}_v$. In fact, the local LPs contain a subset of these constraints. Indeed, for each vertex v, the algorithm defines a subset $\mathcal{R}(v) \subseteq \mathfrak{T}$ of timesteps *relevant* to the vertex v, and adds in a non-empty set of constraints $\mathfrak{L}^v(\tau)$ for each such timestep $\tau \in \mathcal{R}(v)$. Each constraint in $\mathfrak{L}^v(\tau)$ is of the form $\varphi_{\boldsymbol{\tau},v}$ for a monotone vector $\boldsymbol{\tau}$ ending at τ. (Recall from Definition 3 that τ_v must correspond to the request time for some source request $\mathbb{s}_q$ at a leaf in $\mathbb{T}_v$.) Now the local LP $\mathfrak{L}^v$ contains all the constraints $\bigcup_{\tau\in\mathcal{R}(v)}\mathfrak{L}^v(\tau)$, and minimizes

$$\sum_{u\in\mathbb{T}_v\setminus\{v\}}\sum_{\tau'} c_u\, y^v(u,\tau').$$

The set of relevant timesteps, and the constraints added to the local LPs have further structure; to explain this, we need the notion of a $\perp$-constraint:

Definition 4. ($\perp$-constraints) *A truncated constraint $\varphi_{\boldsymbol{\tau},v}$ is called a* $\perp$-constraint *if $\tau_u = \tau_v$ for all $u \in \mathbb{T}_v$.*

It will turn out that for all relevant $\perp$-constraints, the RHS is positive. Therefore, such $\perp$-constraints can never be satisfied, but will nevertheless be useful when forming new constraints by composition.

The timesteps in $\mathcal{R}(v)$ are partitioned into $\mathcal{R}^s(v)$ and $\mathcal{R}^{ns}(v)$, the *solitary* and *non-solitary* timesteps for v. For each timestep $\tau \in \mathfrak{T}$, the algorithm may add τ to either $\mathcal{R}^s(v)$ or $\mathcal{R}^{ns}(v)$ (and hence to $\mathcal{R}(v)$), or it may decide to not add τ to $\mathcal{R}(v)$ at all. For each timestep $\tau \in \mathcal{R}^s(v)$, the algorithm creates a constraint set $\mathfrak{L}^v(\tau)$ consisting of a single $\perp$-constraint (recall Definition 4); for each timestep $\tau \in \mathcal{R}^{ns}(v)$ it creates a constraint set $\mathfrak{L}^v(\tau)$ containing only non-$\perp$-constraints. Each of these constraints is formed by composing constraints from $\mathfrak{L}^w(\tau_w)$ for some children $w \in \chi_v$ and timesteps $\tau_w \in \mathcal{R}(w)$, where $\tau_w \le \tau$.

For each timestep τ, a constraint $C \in \mathfrak{L}^v(\tau)$ gives rise to a dual variable z_C, which is raised only at timestep τ. We ensure the following invariant; recall that γ is the tiny quantity $\frac{1}{n^4}$.

Invariant (I2). *At the end of each timestep $\tau \in \mathcal{R}^{ns}(v)$, the objective function value of the dual variables corresponding to constraints in $\mathfrak{L}^v(\tau)$ equals γ. I.e., if a generic constraint C is given by $\langle a^C \cdot y^v \rangle \geq b^C$, then*

$$\sum_{C \in \mathfrak{L}^v(\tau)} b^C \cdot z_C = \gamma \qquad \forall \tau \in \mathcal{R}^{ns}(v). \tag{I2}$$

Furthermore, $b^C > 0$ for all $C \in \mathfrak{L}^v(\tau)$ and $\tau \in \mathcal{R}(v)$.

No dual variables z_C are defined for $\perp$-constraints, and the first statement of Invariant (I2) does not apply to timesteps $\tau \in \mathcal{R}^s(v)$. In the following sections, we show how to maintain a dual solution that is feasible for $\mathfrak{D}^v$ (the dual LP for $\mathfrak{L}^v$) when scaled down by some factor $\beta = \text{poly}\log(n\lambda)$.

Awake Timesteps. For a vertex v, we maintain a subset $\mathsf{Awake}(v) \subseteq \mathcal{R}(v)$ of *awake* timesteps. The set $\mathsf{Awake}(v)$ has the property that it contains the set $\mathcal{R}^s(v)$ of all the solitary timesteps for v; i.e., $\mathcal{R}^s(v) \subseteq \mathsf{Awake}(v) \subseteq \mathcal{R}^s \cup \mathcal{R}^{ns}(v) = \mathcal{R}(v)$. This set $\mathsf{Awake}(v)$ changes over time. When we add a timestep to $\mathcal{R}(v)$, we also add it to $\mathsf{Awake}(v)$; subsequently, some non-solitary timesteps may be removed from it. A timestep τ is *awake for vertex v* at some moment in the algorithm if it belongs to $\mathsf{Awake}(v)$ at that moment. For any vertex v, define

$$\mathsf{prev}(v, \tau) := \max\{\tau' \in \mathsf{Awake}(v) \mid \tau' \leq \tau\} \tag{3.5}$$

Note that as the set $\mathsf{Awake}(v)$ evolves over time, so does the identity of $\mathsf{prev}(v, \tau)$. We show in Claim 5.2 that prev is well-defined for all relevant (v, τ) pairs.

3.4 The Starting configuration We assume that each leaf node has two copies: one acts as a source node and the other as a sink node. We also add an additional root-to-leaf path to $\mathbb{T}$, where the leaf node (denoted ℓ_{fake}) is at the same level as the other leaf nodes in the tree. We prepend the actual request sequence with $n+1$ additional requests. The first request, called a *fake request*, is $(\ell_{\text{fake}}, \ell_{\text{fake}})$ and appears at time 0. The next n requests corresponds to the n distinct leaves v in $\mathbb{T}$ and are of the form (v, ℓ_{fake}). The node ℓ_{fake} does not get any other request in the rest of the request sequence (i.e., it is not part of the actual request sequence). The utility of these additional requests is that it creates the right set of $\perp$ constraints in $\mathbb{T}$ for the inductive definition of the algorithm. The algorithm does not explicitly serve the fake request; rather, we assume that the configuration of the servers at time 0 is to have $-4\delta n$ servers at each source node and the balance servers at the sink node corresponding to ℓ_{fake}. The subsequent n requests are served by the algorithm as if they were real requests. These transformations increase the optimal solution by at most $O(n\Delta)$, where Δ is the diameter of the HST, while the cost to produce the initial configuration(from any other configuration of servers) is at most $O(k\Delta)$.

4 The Algorithm for k-Taxi

We can now describe our algorithm $\mathcal{A}$ for k-Taxi. At request time q, the request $(\mathbb{s}_q, \mathbb{t}_q)$ arrives. As explained in §3.1,

(i) we first transfer one unit of server mass from $\mathbb{s}_q$ to the holding node $\mathbb{t}'_q$,
(ii) then move mass from other nodes to $\mathbb{s}_q$ (if needed), and
(iii) finally move mass from the holding node $\mathbb{t}'_q$ to the co-located destination node $\mathbb{t}_q$.

The interesting part—and the only part that incurs any cost—is (ii), which is done over a sequence of steps: the *main procedure* (given in Algorithm 1) calls *local update* procedures for each ancestor of $\mathbb{s}_q$. Each such local update moves fractional server mass to $\mathbb{s}_q$ (if possible), and also adds constraints to the local LPs and raises the primal/dual values to account for this movement. We now explain this in detail.

4.1 The Main Procedure In the main procedure of Algorithm 1, let the *backbone* be the leaf-root path $\mathbb{s}_q = v_0, v_1, \ldots, v_H = \mathbb{r}$. We move servers from other leaves to $\mathbb{s}_q$ until it is saturated: this server movement happens in small discrete increments over several timesteps. Each iteration of the **while** loop in line (1.5) corresponds to a distinct timestep τ. Let $\mathsf{activesib}(v, \tau)$ be the siblings v' of v with active leaves in their sub-trees $\mathbb{T}_{v'}$ (at timestep τ). Let i_0 be the smallest index with non-empty $\mathsf{activesib}(v_{i_0}, \tau)$. The procedure SimpleUpdate adds a $\perp$-constraint to each of the sets $\mathfrak{L}^{v_i}(\tau)$ for $i = 0, \ldots, i_0$. For $i > i_0$, the procedure FullUpdate adds (non-$\perp$) constraints to $\mathfrak{L}^{v_i}(\tau)$. If $\mathsf{activesib}(v_i, \tau)$ is non-empty, the procedure also transfers some servers to $\mathbb{s}_q$ from the sub-trees below $\mathsf{activesib}(v_i, \tau)$.

Algorithm 1: Main Procedure

```
1.1  foreach $q = 1, 2, \ldots$ do
1.2      get request $r_q = (\mathbb{s}_q, \mathbb{t}_q)$; let the path from $\mathbb{s}_q$ to the root be $\mathbb{s}_q = v_0, v_1, \ldots, v_H = \mathbb{r}$.
1.3      Create a holding copy $\mathbb{t}'_q$ of $\mathbb{t}_q$ and transfer 1 unit of server mass from $\mathbb{s}_q$ to $\mathbb{t}'_q$.
1.4      $\tau \leftarrow q + \eta$, the first timestep after $q$.
1.5      while $k_{v_0,\tau} \leq -4\delta n$ do
1.6          let $i_0 \leftarrow$ smallest index such that $\mathsf{activesib}(v_{i_0}, \tau) \neq \varnothing$.
1.7          for $i = 0, \ldots, i_0$ do call SimpleUpdate$(v_i, \tau)$.
1.8          for $i = i_0 + 1, \ldots, H$ do call FullUpdate$(v_i, \tau)$.
1.9          $\tau \leftarrow \tau + \eta$.                    // move to the next timestep
1.10     Merge $\mathbb{t}'_q$ with $\mathbb{t}_q$.
```

4.2 The Simple Update Procedure Recall that the SimpleUpdate process is called only for those ancestors of the request node $\mathbb{s}_q$ which do not have active siblings, and hence which do not transfer any server mass to $\mathbb{s}_q$. (Hence they are "solitary" at this time.) This procedure adds timestep τ to the set $\mathcal{R}^s(v)$ of solitary timesteps (and also to $\mathsf{Awake}(v)$), and creates a $\perp$-constraint in the LP $\mathfrak{L}^v$. This constraint can be thought of merely as a book-keeping device.

Algorithm 2: SimpleUpdate(v, τ)

```
2.1  add timestep $\tau$ to the event set $\mathcal{R}^s(v)$ and to $\mathsf{Awake}(v)$.      // "solitary" timestep for $v$
2.2  $\mathfrak{L}^v(\tau) \leftarrow$ the $\perp$-constraint $\varphi_{\tau,v}$, where $\tau_w = \tau$ for nodes $w \in \mathbb{T}_v$.
```

4.3 The Full Update Procedure The FullUpdate(v, τ) procedure is called for backbone nodes v that are above v_{i_0} (using the notation of Algorithm 1). It has two objectives. Firstly, it transfers servers to the requested source leaf node v_0 from the subtrees of the off-backbone children of v, incurring a cost of at most γ. Secondly, it accounts for this movement using the local LPs. Specifically, it defines a set of constraints $\mathfrak{L}^v(\tau)$ and runs a primal-dual update on these constraints until the total dual value raised is exactly γ, i.e., at least the server transfer cost.

We now explain the steps of Algorithm 3 in words. (The notions of slack and depleted constraints will be given in Definition 5. We use $\mathsf{ReqLoc}(\tau)$ to denote the location of the source request at time $\lfloor \tau \rfloor$, i.e., $\mathbb{s}_{\lfloor \tau \rfloor}$.)

Consider a call to FullUpdate(v, τ) with u_0 being the child of v on the path to the source request v_0. Each iteration of the **repeat** loop adds a constraint C to $\mathfrak{L}^v(\tau)$ and raises the dual variable z_C corresponding to it. For each node u in $U := \{u_0\} \cup \mathsf{activesib}(u_0, \tau)$, define $\tau_u := \mathsf{prev}(u, \tau)$ to be the most recent timestep before τ that is currently in $\mathsf{Awake}(u)$. This timestep τ_u may move backwards over the iterations as nodes are removed from $\mathsf{Awake}(u)$ in line (3.17). One exception is the node u_0, for which τ_{u_0} stays equal to τ for the entire run of FullUpdate. Indeed, we would have added τ to $\mathsf{Awake}(u_0)$ (during SimpleUpdate(u_0, τ) or FullUpdate(u_0, τ)) before calling FullUpdate(v, τ), and Claim 5.5 shows that τ stays awake in $\mathcal{R}(u_0)$ during FullUpdate(v, τ).

1. We add constraint $C(v, \sigma, \tau)$ to $\mathfrak{L}^v(\tau)$ by taking one constraint $C_u \in \mathfrak{L}^u(\tau_u)$ for each $u \in U$ and setting $\sigma := (C_{u_0}, \ldots, C_{u_\ell})$. (The choice of constraint from $\mathfrak{L}^u(\tau_u)$ is given in item 3 below.) Each C_u has the form $\varphi_{\boldsymbol{\tau}(u),u}$ for some tuple $(\boldsymbol{\tau}(u))$ ending at $\tau_u := \boldsymbol{\tau}(u)_u$. The new constraint $C(v, \sigma, \tau)$ is its composition $\varphi_{\boldsymbol{\tau},v}$ (see §3.2). For a child u of v, let I_u denote the interval $(\boldsymbol{\tau}_u, \tau_v]$. Observe that the LHS of the constraint $C(v, \sigma, \tau)$ contains variables of the form $y^v(u, t), t \in I_u$ for all children u of v.

2. Having added this constraint $C(v, \sigma, \tau)$, we raise the new dual variable $z_{C(v,\sigma,\tau)}$ at a constant rate in line (3.13), and the primal variables $y^v(u, \tau')$ for each $u \in U$ and any τ' in an index set S_u using an exponential update rule in line (3.15). The index set S_u consists of all timesteps in $I_u \cap \mathcal{R}^{ns}(u)$ and the first timestep of I_u

Algorithm 3: FULLUPDATE(v, τ)

```
3.1  let $h \leftarrow \mathsf{level}(v) - 1$ and $u_0 \in \chi_v$ be child containing the current source request $v_0 := \mathsf{ReqLoc}(\tau)$.
3.2  let $U \leftarrow \{u_0\} \cup \mathsf{activesib}(u_0, \tau)$; say $U = \{u_0, u_1, \ldots, u_\ell\}$, $L_U \leftarrow$ active leaves below $U \setminus \{u_0\}$.
3.3  add timestep $\tau$ to the event set $\mathcal{R}^{ns}(v)$ and to $\mathsf{Awake}(v)$.        // "non-solitary" timestep for $v$
3.4  set timer $s \leftarrow 0$.
3.5  repeat
3.6      for $u \in U$ do
3.7          let $\tau_u \leftarrow \mathsf{prev}(u, \tau)$ and $I_u = (\tau_u, \tau]$.
3.8          let $C_u$ be a slack constraint in $\mathfrak{L}^u(\tau_u)$.        // slack constraint exists since $\mathsf{prev}(u, \tau)$ is awake
3.9      let $\sigma \leftarrow (C_{u_0}, C_{u_1}, \ldots, C_{u_\ell})$ be the resulting tuple of constraints.
3.10     add new constraint $C(v, \sigma, \tau)$ to the constraint set $\mathfrak{L}^v(\tau)$.
3.11     while all constraints $C_{u_j}$ in $\sigma$ are slack and dual objective for $\mathfrak{L}^v(\tau)$ less than $\gamma$ do
3.12         increase timer $s$ at uniform rate.
3.13         increase $z_{C(v,\sigma,\tau)}$ at the same rate as $s$.
3.14         for all $u \in U$, define $S_u := I_u \cap (\mathcal{R}^{ns}(u) \cup \{\tau_u + \eta\})$.
3.15         increase $y^v(u, t)$ for $u \in U, t \in S_u$ according to $\frac{dy^v(u,t)}{ds} = \frac{y^v(u,t)}{\lambda^h} + \frac{\gamma}{Mn \cdot \lambda^h}$.
3.16         transfer server mass from $\mathbb{T}_u$ into $v_0$ at rate $\frac{dy^v(u,I_u)}{ds} + \frac{b^{C_u}}{\lambda^h}$ using the leaves in $L_U \cap \mathbb{T}_u$, for each
               $u \in U \setminus \{u_0\}$
3.17     foreach constraint $C_{u_j}$ that is depleted do
3.18         if all the constraints in $\mathfrak{L}^{u_j}(\tau_{u_j})$ are depleted then remove $\tau_{u_j}$ from $\mathsf{Awake}(u_j)$.
3.19 until the dual objective corresponding to constraints in $\mathfrak{L}^v(\tau)$ becomes $\gamma$.
```

(which is $\tau_u + \eta$ if I_u is non-empty[2]). This index set ensures that S_u is not too large, yet it captures all the "necessary" variables that should be raised (see Figure 1) Moreover, we transfer servers from active leaves in $\mathbb{T}_u$ into $\mathsf{ReqLoc}(q)$ in line (3.16). This transfer is done arbitrarily, i.e., we move servers out of any of the leaf nodes that were active at the beginning of this procedure. Our definition of $\mathsf{activesib}(u_0, \tau)$ means that $\mathbb{T}_u$ has at least one active leaf and hence at least δ servers to begin with. Since we move at most $\gamma \ll \delta$ amounts of server, we maintain Invariant (I1) as shown in Claim 5.8. Since $\tau_{u_0} = \tau$, the interval I_{u_0} is empty so no variables $y^v(u_0, t)$ are raised. Somewhat unusually for an online primal-dual algorithm, the LP variables are only used to account for our algorithm's cost, and not for actual algorithmic decisions (i.e., the server movements). This allows us to increase variables from the past since even though the corresponding server movements are always executed at the current time.

To describe the stopping condition for this process, we first explain the relationship between these local LPs, and the notion of *slack* and *depleted* constraints. We use the fact that for any $u \in U$, we have an almost-feasible dual solution $\{z_C\}_{C \in \mathfrak{L}^u(\tau_u)}$. This in turn corresponds to an increase in primal values for variables $y^u(u', \tau')$ in $\mathfrak{L}^u$. It will suffice for our proof to ensure that when we raise $z_{C(v,\sigma,\tau)}$, we constrain it as follows:

INVARIANT (I1). *For every $u \in \chi_v, t \in \mathcal{R}^{ns}(u)$, and every constraint $C \in \mathfrak{L}^u(t)$ (which by definition of $\mathcal{R}^{ns}(u)$ is not a $\perp$-constraint):*

$$\left(1 + \frac{1}{H}\right) z_C \geq \sum_{\tau' \geq t} \sum_{\sigma : C \in \sigma} z_{C(v,\sigma,\tau')}. \tag{I3}$$

DEFINITION 5. (SLACK AND DEPLETED LOCAL CONSTRAINTS) *A non-$\perp$ constraint $C \in \mathfrak{L}^u$ is* slack *if (I3) is satisfied with a strict inequality, else it is* depleted. *By convention, $\perp$-constraints are always slack.*

[2]Observe that this timestep may not belong to $\mathcal{R}(u)$, but all other timesteps in S_u lie in $\mathcal{R}(u)$ (also see Figure 1).

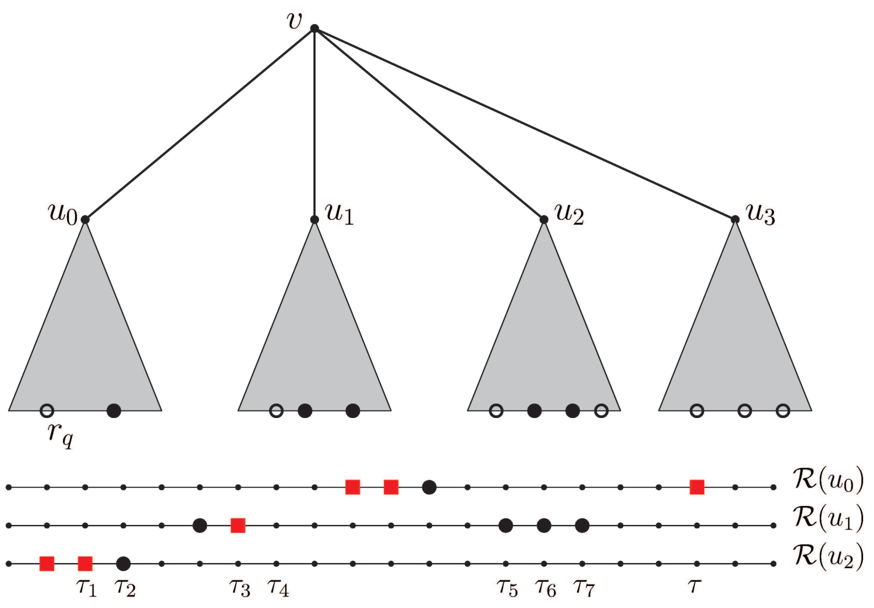

Figure 1: Illustration of $\textsc{FullUpdate}(v,\tau)$*: leaves with filled dots are active and open dots are inactive, so* $\mathsf{activesib}(u_0,\tau) = \{u_1,u_2\}$*. The bold red squares or black circles denote timesteps in* $\mathcal{R}(u) = \mathcal{R}^s(u) \cup \mathcal{R}^{ns}(u)$*, with the red squares being awake at timestep* τ*. Hence,* $I_{u_0} = S_{u_0} = \varnothing$*,* $I_{u_1} = (\tau_3,\tau], S_{u_1} = \{\tau_4,\tau_5,\tau_6,\tau_7\}, I_{u_2} = (\tau_1,\tau], S_{u_2} = \{\tau_2\}$*. Timesteps in* $\mathcal{R}^s(u)$ *always remain awake.*

We can now explain the remainder of the local update.

3. The choice of the constraint in line (3.8) is now easy: C_u is chosen to be any slack constraint in $\mathfrak{L}^u(\tau_u)$. If $\tau_u \in \mathcal{R}^s(u)$, this is the unique $\perp$-constraint in $\mathfrak{L}^u(\tau_u)$.

   The primal-dual update in the **while** loop proceeds as long as all constraints C_u in σ are slack: once a constraint becomes tight, some other slack constraint $C_u \in \mathfrak{L}^u(\tau_u)$ is chosen to be in σ. If there are no more slack constraints in $\mathfrak{L}^u(\tau_u)$, the timestep τ_u is removed from the awake set (in line (3.17)). In the next iteration, τ_u gets redefined to be the most recent awake timestep before τ (in line (3.7)). To argue this is valid, Claim 5.2 shows that there is always an awake timestep on the timeline of every vertex.

4. The dual objective corresponding to constraints in $\mathfrak{L}^v(\tau)$ is $\sum_{C\in\mathfrak{L}^v(\tau)} b^C z_C$, where C is of the form $\langle a^C, y^v\rangle \geq b^C$. The local update process ends when the increase in this dual objective due to raising variables $\{z_C \mid C \in \mathfrak{L}^v(\tau)\}$ equals γ.

For a constraint $C \in \mathfrak{L}^u(t)$, the variable z_C is only raised in the call $\textsc{FullUpdate}(u,t)$. Subsequently, only the right side of (I3) can be raised. Hence, once a constraint C becomes depleted, it stays depleted. It is worth discussing the special case when $\mathsf{activesib}(u_0,\tau)$ is empty, so that $U = \{u_0\}$. In this case, no server transfer can happen. The constraint $C(v,\sigma,\tau)$ is obtained by composition of just one constraint, namely a constraint $C_{u_0} \in \mathfrak{L}^{u_0}(\tau)$. It is easy to check that the LHS of both the constraints is essentially the same (the only difference being that the names of the variables are y^v and y^{u_0} respectively). We shall show that the RHS of $C(v,\sigma,\tau)$ exceeds that of C_{u_0} by at least $(n_v - n_{u_0})\delta$. Our algorithm raises the dual variable $z_{C(v,\sigma,\tau)}$, and we shall show in Claim 5.5 that the dual objective value rises by γ.

There is a parameter M in line (3.15) that specifies the rate of change of y^v. This value M should be an upper bound on the size of the index set S_u for all calls to $\textsc{FullUpdate}$, and over all $u \in U$. We will set the value of M to $\frac{5H\lambda^H(T+k)}{4\gamma} + 1$ in Corollary 5.3, where T is the length of time horizon (i.e., number of requests in the input). Using this value of M will cause the competitive ratio to depend on $\log T$; we shall show in §5.4 how to remove this dependence on T.

5 Analysis Details

Now that we have chosen the "right" LP relaxation, i.e., the "right" set of constraints to consider, we can lean on the analytic machinery from [GKP21]. Indeed, the proofs of this section closely mirror those from [GKP21]; we give the entire proof here for completeness. Among the crucial new ingredients is our approach to control the size of the linear program, which we present in §5.4.

The proof relies on two lemmas: the first (proved in §5.2) bounds the movement cost in terms of the increase in dual value, and the second (proved in §5.3) shows near-feasibility of the dual solutions.

Lemma 5.1. (Server Movement) *The total movement cost during an execution of the procedure* FullUpdate *is at most 2γ, and the objective value of the dual $\mathfrak{D}^v$ increases by exactly γ.*

Lemma 5.2. (Dual Feasibility) *For each vertex v, the dual solution to $\mathfrak{L}^v$ is β-feasible, where $\beta = O(\log \frac{nMk}{\gamma}) = O(H \log(nk\lambda T))$.*

The last equation follows from substituting the value of $M = \frac{5H\lambda^H(T+k)}{4\gamma} + 1$ (from Corollary 5.3) and $\gamma = 1/n^4$ (defined before Definition 2). We also use the fact that $H < \lambda$ (see the first paragraph in Section 2.1). Later, in Section 5.4, we improve our bound by removing the dependence on T in the competitive ratio.

Theorem 5.1. (Competitiveness for k-Taxi) *Given any instance of the k-Taxi problem on a λ-HST with height at most $\lambda/10$, Algorithm 1 ensures that each request location $\mathfrak{s}_q$ is saturated at some timestep in $[q, q+1)$. The total cost of (fractional) server movement is $O(\beta H) = O(H^2 \log(n\lambda T))$ times the cost of the optimal solution.*

Proof. All the server movement happens within calls to FullUpdate. By Lemma 5.1, each iteration of the **while** loop of line (1.5) in Algorithm 1 incurs a total movement cost of $O(H\gamma)$ over at most H invocations. Moreover, the call FullUpdate($\mathfrak{r}, \tau$) increases the value of the dual solution to the LP $\mathfrak{L}^{\mathfrak{r}}$ by γ. This means the total movement cost is at most $O(H)$ times the dual solution value. Since all constraints of $\mathfrak{L}^{\mathfrak{r}}$ are implied by the relaxation $\mathfrak{M}$, any feasible dual solution gives a lower bound on the optimal solution to $\mathfrak{M}$. By Lemma 5.2, our dual solution is feasible if it is scaled down by β, and so we get that our (fractional) algorithm is $O(\beta H) = O(H^2 \log(n\lambda T))$-competitive. ∎

The competitive ratio in Theorem 5.1 depends on the length T of the time horizon. We remove this dependence on T by dividing the the input in phases and running Algorithm 1 independently in each phase. The details of this algorithm are given in §5.4.

Using a standard metric embedding technique to embed λ-HSTs with $\lambda = O(\log \Delta)$ allows us to extend this result to general metrics with a further loss of $O(\lambda \log_\lambda(n\Delta)) = O(\log \Delta \log(n\Delta))$.

5.1 Rewriting Local Constraints in Terms of Local Changes In this section, we rewrite the local constraints (3.4). Recall that if X is a subset of children of a node v, and we given local constraints C_u ending at a timestep τ_u for each $u \in X$, then we can compose these constraints to get a local constraint at v (ending at a suitably defined time τ_v). We now express this local constraint at v in terms of the server movement that happens between v and each child $u \in X$ during $(\tau_u, \tau_v]$. We begin by defining this notion of server movement.

Definition 6. (g, r, D) *For a vertex v and timestep t, let "give" $g(v,t)$ and "receive" $r(v,t)$ denote the total (fractional) server movement* out of *and* into *the subtree $\mathbb{T}_v$ on the edge $(v, p(v))$ at timestep t. For an interval I, let $g(v,I) := \sum_{t \in I} g(v,t)$ and define $r(v,I)$ similarly, and let "difference" $D(v,I) := g(v,I) - r(v,I)$.*

We now show how the difference term D can be used to account for change of server mass in a subtree.

Claim 5.1. *Consider a node u in $\mathbb{T}$ and let v be the parent of u. Let $\boldsymbol{\tau}$ be a monotone vector at v and let $\boldsymbol{\tau}(u0$ be the restriction of this vector to the subtree $\mathbb{T}_u$. Assume that $\lfloor s_{\tau_v} \rfloor \notin \mathbb{T}_u$. Then,*

$$k_{u,\tau_2} - k_{u,\tau_1} = -D(u,(\tau_1,\tau_2]) + |P_v(\boldsymbol{\tau}) \cap \mathbb{T}_u| - |P_u(\boldsymbol{\tau}(u))| - |N_v(\boldsymbol{\tau})| + |N_u(\boldsymbol{\tau}(u))|.$$

Proof. Let I denote the interval $(\tau_1, \tau_2]$. Let v be the parent of u. The server mass entering $\mathbb{T}_u$ during I using the edge (u,v) is equal to $-D(u,I)$. Now, for each request $q \in I$ one unit of server mass enters $\mathbb{T}_u$ if $t_q \in \mathbb{T}_u$. The total such server mass entering $\mathbb{T}_u$ during I is equal to $|P_v(\boldsymbol{\tau}) \cap \mathbb{T}(u)| - |P_u(\tau(u))|$. Indeed, consider such a request q with $t_q \in \mathbb{T}_u$. Then $\tau_u < \tau_{t_q} \le \tau_v$, and so, $t_q \in P_v(\boldsymbol{\tau}) \setminus P_u(\boldsymbol{\tau}(u))$. Conversely, if $t_q \in P_v(\boldsymbol{\tau}) \cap \mathbb{T}_u \setminus P_u(\boldsymbol{\tau}(u))$, then the request q arrives during I. Finally, for each request $q \in I$, one unit of server mass leaves $\mathbb{T}_u$ if $s_q \in \mathbb{T}_u$. Using a similar argument as above, this latter server mass movement is captured by $|N_v(\boldsymbol{\tau})| - |N_u(\boldsymbol{\tau}(u))|$, except the case when $q = \lfloor \tau_2 \rfloor$ (this is because of the definition of $N_v(\boldsymbol{\tau})$ which does not count the request at time $\lfloor \tau \rfloor$). However, we know that $s_{\lfloor \tau_2 \rfloor} \notin \mathbb{T}_u$, and so we need not consider this case. This proves the desired result. ∎

These difference terms D allow us to write down a suitable local constraint (5.6) implied by $\varphi_{\boldsymbol{\tau},v}$.

LEMMA 5.3. *Suppose we are given a vertex v, a timestep τ and a subset X of children of v such that at timestep τ all active leaves in $\mathbb{T}_v$ are descendants of one of the nodes in X. For each $u \in X$, we are also given a truncated constraint $C_u := \varphi_{\boldsymbol{\tau}(u),u}$ specified by some linear inequality $\langle a^{C_u}, y^u \rangle \ge b^{C_u}$. Define $\tau_v := \max_{u \in X} \tau_u$, and let $\varphi_{\boldsymbol{\tau},v}$ be the local constraint (ending at τ_v) in $\mathfrak{L}^v$ obtained by composition of the $C_u, u \in X$. Assume that $\tau = \tau_v$ and X is non-empty. Then the constraint $\varphi_{\boldsymbol{\tau},v}$ implies the inequality*

$$\sum_{u \in X} \Big(y^v(u, (\tau_u, \tau_v]) + \langle a^{C_u}, y^v \rangle \Big) \ge \sum_{u \in X} \Big(D(u, (\tau_u, \tau_v]) + b^{C_u} \Big) + \big(n_v - \sum_{u \in X} n_u\big)\delta, \tag{5.6}$$

This implied constraint (5.6) should be read as follows: during any time interval, either the constraints $\langle a^{C_u}, y^u \rangle \ge b^{C_u}$ at the children are satisfied (and hence have generated dual), or else the algorithm's movement of servers between subtrees (which shows up in the $D(\cdot,\cdot)$ on the RHS) is reflected in the y^v variables on the edges from v to its children (and hence also generate dual). The proof of Lemma 5.3 is just careful algebraic manipulation, and can be skipped at the first reading.

Proof. The LHS of $\varphi_{\boldsymbol{\tau},v}$ is the same as that of (5.6). It remains to show that the RHS of $\varphi_{\boldsymbol{\tau},v}$ is at least that of (5.6). In other words, we need to show that

$$\begin{aligned} |P_v(\boldsymbol{\tau})| - |N_v(\boldsymbol{\tau})| - k_{v,\tau_v} - 2\delta(n - n_v) &\ge \sum_{u \in X} \Big(D(u, (\tau_u, \tau_v]) + b^{C_u} \Big) + \big(n_v - \sum_{u \in X} n_u\big)\delta \\ &= \sum_{u \in X} \Big(D(u, (\tau_u, \tau_v]) + |P_u(\boldsymbol{\tau}(u))| - |N_u(\boldsymbol{\tau}(u))| - k_{u,\tau_u} - 2\delta(n - n_u) \Big) + \big(n_v - \sum_{u \in X} n_u\big)\delta \end{aligned}$$

Since $|X| \ge 1$, the above inequality is implied by the following:

$$|P_v(\boldsymbol{\tau})| - |N_v(\boldsymbol{\tau})| - k_{v,\tau_v} \ge \sum_{u \in X} \Big(D(u, (\tau_u, \tau_v]) + |P_u(\boldsymbol{\tau}(u))| - |N_u(\boldsymbol{\tau}(u))| - k_{u,\tau_u} \Big) - \big(n_v - \sum_{u \in X} n_u\big)\delta$$

Rearranging terms, the above is same as

$$\begin{aligned} k_{v,\tau_v} - \sum_{u \in X} k_{u,\tau_u} \le - \sum_{u \in X} D(u, (\tau_u, \tau_v]) &+ \Big(|P_v(\boldsymbol{\tau})| - \sum_{u \in X} |P_u(\boldsymbol{\tau}(u))| \Big) - \Big(|N_v(\boldsymbol{\tau})| - \sum_{u \in X} |N_u(\boldsymbol{\tau}(u))| \Big) \\ &+ \big(n_v - \sum_{u \in X} n_u\big)\delta \end{aligned} \tag{5.7}$$

Note that in the last equality above, the term for $u = u_0$ is 0. In rest of the proof, we show (5.7). Since $\tau_v = \max_{u \in X} \tau_u$, let $u_0 \in X$ be such that $\tau_v = \tau_{u_0}$. By definition of monotone timestep vector, we know that the request $s_q, q = \lfloor \tau_v \rfloor$, is a leaf in $\mathbb{T}_{u_0}$. Thus we can apply Claim 5.1 to any node $u \in X \setminus \{u_0\}$ for the interval $(\tau_u, \tau_v]$. We get:

$$\begin{aligned} k_{v,\tau_v} - \sum_{u \in X} k_{u,\tau_u} &= \sum_{u \in X \setminus \{u_0\}} (k_{u,\tau_v} - k_{u,\tau_u}) + \sum_{u \in \overline{X}} k_{u,\tau_v} \\ &= \sum_{u \in X} \Big(- D(u, (\tau_u, \tau_v]) + |P_v(\boldsymbol{\tau}) \cap \mathbb{T}_u| - |P_u(\boldsymbol{\tau}(u))| - |N_v(\boldsymbol{\tau}) \cap \mathbb{T}_u)| + |N_u(\boldsymbol{\tau}(u))| \Big) + \sum_{u \in \overline{X}} k_{u,\tau_v}. \end{aligned} \tag{5.8}$$

Let q denote $\lfloor \tau_v \rfloor$. Now observe that $P_v(\boldsymbol{\tau})$ is same as $\cup_{u\in X}(P_v(\boldsymbol{\tau}) \cap \mathbb{T}_u)$, except perhaps the requested location t_q (this is because $\tau_w = \tau_v$ for all nodes in $\mathbb{T}_{u'}, u' \in \chi(v) \setminus X$). Since s_q does not get counted in $N_v(\boldsymbol{\tau})$ (or $N_u(\boldsymbol{\tau}(u))$ for any child u of v), $N_v(\boldsymbol{\tau})$ is equal to $\cup_{u\in X}(N_v(\boldsymbol{\tau}) \cap \mathbb{T}_u)$.

Thus, there are three possible cases based on the location of the sink request t_q. The first case is when t_q lies below one of the nodes in X. Then the above expression equals:

$$-\sum_{u\in X} D(u,(\tau_u,\tau_v]) + \left(|P_v(\boldsymbol{\tau})| - \sum_{u\in X}|P_u(\boldsymbol{\tau}(u))|\right) - \left(|N_v(\boldsymbol{\tau})| - \sum_{u\in X}|N_u(\boldsymbol{\tau}(u))|\right) + \sum_{u\in\overline{X}} k_{u,\tau_v}$$

This implies (5.7) because for a node $u \in \overline{X}$, $k_{u,\tau_v} \le n_u\delta$ and so

$$\sum_{u\in\overline(X)} k_{u,\tau_v} \le \left(n_v - \sum_{u\in X} n_u\right)\delta.$$

The second case happens when the sink request at time $\lfloor \tau_v \rfloor$ lies below a node $u' \in \overline{X}$. In this case, as argued above, (5.8) simplifies to

$$-\sum_{u\in X} D(u,(\tau_u,\tau_v]) + \left(|P_v(\boldsymbol{\tau})| - \sum_{u\in X}|P_u(\boldsymbol{\tau}(u))| - 1\right) - \left(|N_v(\boldsymbol{\tau})| - \sum_{u\in X}|N_u(\boldsymbol{\tau}(u))|\right) + \sum_{u\in\overline{X}} k_{u,\tau_v}.$$

Not counting the holding node for request q, we know that for all $u \in \overline{X}$, $k_{u,\tau_u} \le n_u\delta$. Thus,

$$\sum_{u\in\overline{X}} k_{u,\tau_v} \le \delta(n_v - \sum_{u\in X} n_u) + 1.$$

This shows that (5.7) holds in this case as well.

The final case is that the sink of the request at time $\lfloor \tau_v \rfloor$ is outside the subtree $\mathbb{T}_v$. This case is identical to the first case described above. ◻

A direct consequence of Equation (5.6) is the following (we use the notation used in Algorithm 3):

Corollary 5.1. *Consider the constraint $C(v,\sigma,\tau)$ added to $\mathfrak{L}^v(\tau)$ line 3.10 in Algorithm 3. Then $C(v,\sigma,\tau)$, given by $a^{C(v,\sigma,\tau)} \cdot y^v \ge b^{C(v,\sigma,\tau)}$, implies the constraint:*

$$\underbrace{\sum_{u\in U}\left(y^v(u,I_u) + a^{C_u}\cdot y^v\right)}_{=a^{C(v,\sigma,\tau)}\cdot y^v} \ge \underbrace{\sum_{u\in U}\left(D(u,I_u) + b^{C_u}\right) + (n_v - \sum_{u\in U} n_u)\delta}_{\le b^{C(v,\sigma,\tau)}}. \tag{5.9}$$

Equipped with these constraints rewritten in terms of the differences, we can return to proving Lemmas 5.1 and 5.2.

5.2 Bounds on Server Transfer and Dual Increase To prove Lemma 5.1, first note that the dual increase of γ in each call to FullUpdate follows from Invariant (I2) (which still needs to be shown). To prove the upper bound on the server movement, we state another invariant below. Then in §5.2 we show both invariants are indeed maintained throughout the algorithm.

We first define the notion of the "lost" dual increase. Consider a call to FullUpdate(v,τ). Let u_0 be v's child such that request location v_0 lies in $\mathbb{T}_{u_0}$. We say that u_0 is v's *principal* child at timestep τ. We prove (in Claim 5.5) that $\tau \in \mathcal{R}(u_0)$ remains in the awake set and hence $\tau_{u_0} = \tau$ throughout this procedure call. The dual update raises $z_{C(v,\sigma,\tau)}$ in line (3.13) and transfers servers from subtrees $\mathbb{T}_{u'}$ for $u' \in \mathsf{activesib}(u_0,\tau)$ into subtree $\mathbb{T}_{u_0}$ in line (3.16). This transfer has two components, which we consider separately. The first is the *local component* $\frac{dy^v(u',t)}{ds}$, and the second is the *inherited component* $\frac{b^{C_{u'}}}{\lambda^h}$. In a sense, the inherited component matches the dual increase corresponding to the term $\sum_{u'\in\mathsf{activesib}(u,\tau)} b^{C_{u'}}$ on the RHS of (5.9). The only term without a corresponding server transfer is $b^{C_{u_0}}$ itself, where C_{u_0} is the constraint in σ corresponding to $\mathfrak{L}^{u_0}(\tau)$. Motivated by this, we give the following definition.

Definition 7. (Loss) *For vertex u_0 with parent v, consider a timestep $\tau \in \mathcal{R}^{ns}(v)$ such that $\tau \in \mathcal{R}(u_0)$ as well. If $\tau \in \mathcal{R}^s(u_0)$, define $\mathsf{loss}(u_0, \tau) := 0$. Else $\tau \in \mathcal{R}^{ns}(u_0)$, and then*

$$\mathsf{loss}(u_0, \tau) := \sum_{C \in \mathfrak{L}^{u_0}(\tau)} \sum_{C(v,\sigma,\tau): C \in \sigma} b^C \; z_{C(v,\sigma,\tau)} \quad . \tag{5.10}$$

Invariant (I1). *For node v and timestep $\tau \in \mathcal{R}^{ns}(v)$, let u_0 be v's principal child at timestep τ. The server mass entering subtree $\mathbb{T}_{u_0}$ (via the edge $(p(u_0), u_0))$) during the procedure* FullUpdate(v, τ) *is at most*

$$\frac{\gamma - \mathsf{loss}(u_0, \tau)}{\lambda^{\mathsf{level}(u_0)}} \tag{I4}$$

Moreover, timestep $\tau \in \mathcal{R}(u_0)$ stays awake during FullUpdate(v, τ).

Multiplying the amount of transfer by the cost of this transfer, we get that the total movement cost is at most $O(\gamma)$. Invariants (I2) and (I1) prove Lemma 5.1. We now show these invariants hold over the course of the algorithm.

5.2.1 Proving Invariants (I2) and (I1) To prove these invariants, we define a total order on pairs (v, τ) with $\tau \in \mathcal{R}(v)$ as follows:

$$\text{define: } (v_1, \tau_1) \prec (v_2, \tau_2) \text{ if } \tau_1 < \tau_2 \text{ or if } \tau_1 = \tau_2 \text{ and } v_1 \text{ is a descendant of } v_2.$$

Since calls to FullUpdate are made in this order, we also prove the invariants by induction on this ordering: Assuming both of them hold for all pairs $(v, \tau) \prec (v^\star, \tau^\star)$, we prove them for the pair $(v^\star, \tau^\star)$. For the base case, we only have to consider the $\perp$-constraints at the leaf nodes at time 0. The only non-trivial statement among Invariants (I2) and (I1) for these nodes is to check that $b^C > 0$ for any such $\perp$-constraint C at a source leaf v. Note that $b^C = 1 - k_{v,0} - 2\delta(n-1) > 0$, since $|N_v(0)| = 0, |P_v(0) = 1|, k_{v,0} = -4\delta n + \frac{4\delta n + k}{n} \leq 1/2$. We first show that the notion of prev timestep in the FullUpdate procedure is well-defined.

Claim 5.2. *Let u be any non-root vertex. Then the first timestep in $\mathcal{R}(u)$ corresponds to a $\perp$-constraint. Therefore, for any timestep τ such that $\mathbb{T}_u$ has an active leaf at timestep τ, $\mathsf{prev}(u, \tau)$ is well defined.*

Proof. Let u be a node as above. If u happens to be the holding node, then the above condition holds by construction. So assume u is not a holding node. The first request q below u has the source leaf $\mathbb{s}_q$ located below u. This is because by Section 3.4, we ensure that the first request at each node v in $\mathbb{T}$ is of the form (v, ℓ_{fake}) where ℓ_{fake} is the fake node defined in Section 3.4. Let τ_f be the first timestep after the arrival of this request. In the first iteration of the **while** loop in Algorithm 1 (corresponding to timestep τ_f), we would call SimpleUpdate(u, τ_f) because there are no active leaves below u at this timestep. Hence we would add a $\perp$-constraint at timestep τ_f, proving the first part of the claim. To show the second part, let τ be a timestep such that $\mathbb{T}_u$ has an active leaf below it at timestep τ. This means that $\tau \geq \tau_f$. Since $\mathfrak{L}^u(\tau_f)$ is a $\perp$-constraint, τ_f is awake, and so $\mathsf{prev}(u, \tau)$ is well-defined. □

We start off with some supporting claims before proving the inductive step Invariants (I2) and (I1).

Definition 8. (fill) *Given a node u and its parent v, timestep $\tau \in \mathcal{R}^{ns}(u)$, and constraint $C \in \mathfrak{L}^u(\tau)$, define $\mathsf{fill}(C)$ to be the timesteps τ' (note that all these timesteps are after τ, i.e., $\tau' \geq \tau$) such that some constraint $C' \in \mathfrak{L}^v(\tau')$ appears on the RHS of inequality (I3) corresponding to C. Extending this, let*

$$\mathsf{fill}(u, \tau) := \bigcup_{C \in \mathfrak{L}^u(\tau)} \mathsf{fill}(C). \tag{5.11}$$

In other words, $\mathsf{fill}(u, \tau)$ is the set of timesteps $\tau' \in \mathcal{R}^{ns}(v)$ such that when we called FullUpdate(v, τ'), node u was either v's principal child at timestep τ' or else belonged to the active sibling set, and moreover $\mathsf{prev}(u, \tau') = \tau$. These are the set of timesteps that *load* constraints in $\mathfrak{L}^u(\tau)$: the following lemma shows part of their structure. (We use $(v^\star, \tau^\star)$ to denote the current pair in the inductive step for proving Invariants (I2) and (I1).)

Claim 5.3. (Structure of fill times) *Fix a node u with parent v, and a timestep $\tau \in \mathcal{R}^{ns}(u)$ such that $(v,\tau) \prec (v^\star,\tau^\star)$. Then for any $\tau' < \tau^*$ such that $\tau' \in \mathsf{fill}(u,\tau)$, either (a) $\tau' = \tau$, in which case u is the principal child of v at timestep τ', or else (b) $\tau' > \tau$, and u is* not *v's principal child at timestep τ'.*

Proof. Suppose $\tau = \tau'$. Since we call FullUpdate only for ancestors of the requested node v_0, and $\tau \in \mathcal{R}^{ns}(u)$, so v_0 belongs to $\mathbb{T}_u$ (and hence u is the principal child of v at timestep τ). Else suppose $\tau' > \tau$, and suppose u is indeed v's principal child at this timestep. Then during the call FullUpdate(v,τ'), we have $\mathsf{prev}(u,\tau') = \tau'$ throughout the execution of FullUpdate(v,τ') (by the second statement in Invariant (I1)), and hence $\tau' \notin \mathsf{fill}(u,\tau)$, giving a contradiction. ∎

We now give an upper bound on the server mass entering a subtree at any timestep $\tau < \tau^\star$.

Claim 5.4. *Let $\tau \in \mathcal{R}(u)$, $\tau < \tau^\star$. The server mass entering $\mathbb{T}_u$ (via the edge $(p(u),u)$) at timestep τ is at most*

$$\left(1+\frac{1}{\lambda-1}\right)\frac{\gamma}{\lambda^{\mathsf{level}(u)}} - \frac{\mathsf{loss}(u,\tau)}{\lambda^{\mathsf{level}(u)}}.$$

Proof. Since $\tau < \tau^\star$, we can apply the induction hypothesis to all pairs (v,τ) where v is an ancestor of u. Servers enter u at timestep τ because of FullUpdate(w,τ) for some ancestor w of u. When w is the parent of u, Invariant (I1) shows this quantity is at most $\frac{\gamma-\mathsf{loss}(u,\tau)}{\lambda^h}$, where $h = \mathsf{level}(u)$. For any other ancestor w of v, we can use a weaker upper bound (again using Invariant (I1)) of $\frac{\gamma}{\lambda^{h+r}}$, where $\mathsf{level}(w) = h+r+1$. Simplifying the resulting geometric sum $\frac{\gamma-\mathsf{loss}(u,\tau)}{\lambda^h} + \sum_{h'\geq h+1}\frac{\gamma}{\lambda^{h'}}$ completes the proof. ∎

Next, we give a lower bound on the amount of server *moving out* of some subtree $\mathbb{T}_w$ via the edge $(w,p(w))$. Recall that all this transfer takes place in line (3.16) with w being either the node u on this line, or a descendant, and that the amount of server movement out of $\mathbb{T}_w$ via the edge $(w,p(w))$ at timestep τ is denoted $g(w,\tau)$, which is non-zero only for those timesteps τ when w is not on the corresponding backbone. We split this transfer amount into two:

(i) $g^{\mathsf{loc}}(w,\tau)$: the *local component* of the transfer, i.e., due to the increase in y^v variables.
(ii) $g^{\mathsf{inh}}(w,\tau)$: the *inherited component* of the transfer, i.e., due to the b^{C_u} term.

Lemma 5.4. *Let u be a non-principal child of $v^\star$ at timestep $\tau^\star$, and $I := (\tau_1,\tau^\star]$ for some timestep $\tau_1 < \tau^\star$. Let S be the timesteps in $\mathcal{R}^{ns}(u)\cap(\tau_1,\tau^\star]$ that have been removed from $\mathsf{Awake}(u)$ by the time that FullUpdate$(v^\star,\tau^\star)$ is called. Then*

$$g^{\mathsf{inh}}(u,(\tau_1,\tau^\star]) \geq \left(1+\frac{1}{H}\right)|S|\frac{\gamma}{\lambda^{\mathsf{level}(u)}} - \sum_{\tau\in S}\frac{\mathsf{loss}(u,\tau)}{\lambda^{\mathsf{level}(u)}}.$$

Proof. Consider timesteps $\tau \in S$ and $\tau' \geq \tau$ such that $\tau' \in \mathsf{fill}(u,\tau)$. Consider the "phase" during FullUpdate$(v^\star,\tau')$ when $\tau'_u := \mathsf{prev}(u,\tau')$ equals τ: since $\tau' \in \mathsf{fill}(u,\tau)$, we know that there will be such a phase. (We use the term phase here informally, to denote a range of values of the timer s.) Whenever we raise the timer s by a small ϵ amount during this phase, we raise some dual variable $z_{C(v^\star,\sigma,\tau')}$ by the same amount, where σ contains a constraint C from $\mathfrak{L}^u(\tau)$. Thus we contribute ϵ to the LHS of (I3) for constraint C. For such a constraint C, let $[\mathfrak{s}_1(\tau',C),\mathfrak{s}_2(\tau',C)]$ be the range of the timer s during which we raise a dual variable of the form $z_{C(v^\star,\sigma,\tau')}$ such that $C\in\sigma$.

If τ has been removed from $\mathsf{Awake}(u)$ (in line (3.17)), it means that (I3) is tight for all constraints $C \in \mathfrak{L}^u(\tau)$, so:

$$\left(1+\frac{1}{H}\right)\sum_{C\in\mathfrak{L}^u(\tau)} b^C z_C = \sum_{C\in\mathfrak{L}^u(\tau)} b^C \sum_{C(v^\star,\sigma,\tau'):C\in\sigma} z_{C(v^\star,\sigma,\tau')}$$

(Note that the above sum is being taken across different τ'.) Now the definition of $\mathsf{loss}(u,\tau)$ allow us to split the expression on the RHS as follows:

$$\left(1+\frac{1}{H}\right)\sum_{C\in\mathfrak{L}^u(\tau)} b^C z_C = \mathsf{loss}(u,\tau) + \sum_{C\in\mathfrak{L}^u(\tau)} b^C \sum_{C(v^\star,\sigma,\tau'):C\in\sigma,\tau'>\tau} z_{C(v^\star,\sigma,\tau')}$$

$$(5.12) \qquad = \mathsf{loss}(u,\tau) + \sum_{\tau' \in \mathsf{fill}(u,\tau), \tau' > \tau} \sum_{C \in \mathfrak{L}^u(\tau)} b^C \Big(\mathbb{s}_2(\tau', C) - \mathbb{s}_1(\tau', C) \Big).$$

We now bound the second expression on the RHS in another way. For a timestep $\tau' \in \mathsf{fill}(u,\tau)$ with $\tau' > \tau$, consider the phase when timer s lies in the range $[\mathbb{s}_1(\tau', C), \mathbb{s}_2(\tau', C)]$ for a constraint $C \in \mathfrak{L}^u(\tau)$. Since $\tau' > \tau$, Claim 5.3 implies that u is not the principal child of $v^\star$ at timestep τ', so raising s by ϵ units during this phase means that line (3.16) moves at least $\epsilon \cdot \frac{b^C}{\lambda^h}$ servers *out* of $\mathbb{T}_u$, where $h := \mathsf{level}(u)$. Hence the increase in $g^{\mathsf{inh}}(u, I)$ due to transfers corresponding to timestep $\tau \in S$ is at least

$$\sum_{\tau' \in \mathsf{fill}(u,\tau), \tau' > \tau} \sum_{C \in \mathfrak{L}^u(\tau)} \frac{b^C(\mathbb{s}_2(\tau', C) - \mathbb{s}_1(\tau', C))}{\lambda^h} \overset{\text{by (5.12)}}{=} \left(1 + \frac{1}{H}\right) \sum_{C \in \mathfrak{L}^u(\tau)} \frac{b^C z_C}{\lambda^h} - \frac{\mathsf{loss}(u,\tau)}{\lambda^h}$$
$$= \left(1 + \frac{1}{H}\right) \frac{\gamma}{\lambda^h} - \frac{\mathsf{loss}(u,\tau)}{\lambda^h}.$$

Since τ was removed from $\mathsf{Awake}(u)$ by the time that FULLUPDATE$(v^\star, \tau^\star)$ is called, the former tuple (u, τ) must strictly precede the latter. Hence, we can use the induction hypothesis (Invariant (I2)) for timestep τ in $\mathcal{R}^{ns}(u)$ to get the second equality above. Finally, summing over all timesteps in S completes the proof. ∎

COROLLARY 5.2. *Let u be a non-principal child of $v^\star$ at timestep $\tau^\star$, and $I := (\tau_1, \tau^\star]$ for any $\tau_1 < \tau^\star$. Suppose none of the timesteps in $I \cap \mathcal{R}(u)$ belong to $\mathsf{Awake}(u)$ when* FULLUPDATE$(v^\star, \tau^\star)$ *is called. Then at this moment,*

(i) $g^{\mathsf{inh}}(u, I) \geq r(u, I)$,

(ii) $0 \leq g^{\mathsf{loc}}(u, I) \leq D(u, I)$, *and*

(iii) $g^{\mathsf{loc}}(u, I) \geq y^{v^\star}(u, I)$.

Finally, $b^C > 0$ for any constraint C of the form $\langle a^C \cdot y^v \rangle \geq b^C$ in $\mathfrak{L}^{v^\star}(\tau^\star)$.

Proof. Since timesteps in $\mathcal{R}^s(u)$ always stay awake, $I \cap \mathcal{R}(u) = I \cap \mathcal{R}^{ns}(u)$; call this set S. Since u is a non-principal child at timestep $\tau^\star$, we have $\tau^\star \notin \mathcal{R}^{ns}(u)$. This means $\tau < \tau^\star$ for any $\tau \in S$, and so Claim 5.4 gives an *upper bound* on the server movement *into* u at timestep τ, and Lemma 5.4 gives a *lower bound* on the server movement *out of* u. Combining the two,

$$(5.13) \qquad g^{\mathsf{inh}}(u, I) - r(u, I) \geq \left(\frac{1}{H} - \frac{1}{\lambda - 1}\right) \frac{\gamma |S|}{\lambda^h} \geq \frac{4}{5H} \cdot \frac{\gamma |S|}{\lambda^h} \geq 0,$$

since $\lambda \geq 10H$ and $H \geq 2$, which proves (i). To prove (ii),

$$g^{\mathsf{loc}}(u, I) = (g - g^{\mathsf{inh}})(u, I) \overset{\text{by (i)}}{\leq} (g - r)(u, I) \overset{\text{by defn.}}{=} D(u, I).$$

To prove (iii), whenever we raised $y^{v^\star}(u, \tau')$ for some timestep τ', we transferred servers out of u (and therefore raised $g(u, \tau'')$ for some $\tau'' \geq \tau'$) with at least as large a rate (since the former only accounts for the local component of the transfer). Both τ', τ'' appeared before $\tau^\star$, because we are at the time that we call FULLUPDATE$(v^\star, \tau^\star)$. Since interval I ends at $\tau^\star$, it must contain either only τ'' or both τ', τ'', giving us that $g^{\mathsf{loc}}(u, I) \geq y^{v^\star}(u, I)$.

We now prove the final statement. We assume that $\lfloor \tau^\star \rfloor > 0$, since the base case has been considered already. If we add a $\perp$- constraint C to $\mathfrak{L}^{v^\star}(\tau^\star)$, then we would have called SIMPLEUPDATE$(v^\star, \tau^\star)$, where v_0 is the requested source node at time $\lfloor \tau^\star \rfloor$. The $\perp$ constraint C at $v^\star$ is given by $\varphi_{v^\star, \boldsymbol{\tau}}$, where $\boldsymbol{\tau}$ assigns timestep $\tau^\star$ to each node in $\mathbb{T}_{v^\star}$. Let v_0' be the holding sink node corresponding to this request. Two cases arise depending on whether v_0' lies in $\mathbb{T}_{v^\star}$. First assume that $v_0' \notin \mathbb{T}_{v^\star}$. Then $P_{v^\star}(\boldsymbol{\tau}) = N_{v^\star}(\boldsymbol{\tau}) = \varnothing$, $k_{v_0, \tau^\star} \leq -4\delta n$ and every sink node in $\mathbb{T}_{v^\star}$ has at most δ server mass. Therefore $b^C = -k_{v^\star, \tau^\star} - 2\delta(n - n_{v^\star}) \geq 4\delta n - \delta n - 2\delta n > 0$. The other case when $v_0' \in \mathbb{T}_{v^\star}$ is similar – the only change is that $k_{v_0', \tau^\star}$ is 1, but then $|P_{v^\star}(\boldsymbol{\tau})|$ is also 1.

Finally consider the case when C is of the form $C(v^\star, \sigma, \tau^\star)$ as in (5.9). By the induction hypothesis (Invariant (I2)), $b^{C_u} > 0$ and $D(u, I_u) \geq 0$ by (ii) above. Since $n_{v^\star} > \sum_{u \in U} n_u$, it follows that $b^C > 0$. ∎

Having proved all the supporting claims, we start off with proving that the second statement in Invariant (I2) holds at $(v^\star, \tau^\star)$.

Claim 5.5. (Principal Node Awake) *Suppose we call* FullUpdate$(v^\star, \tau^\star)$. *If u is the principal child of $v^\star$ at timestep $\tau^\star$, this call does not remove the timestep $\tau^\star$ from* $\mathsf{Awake}(u)$.

Proof. At the beginning of the call to FullUpdate$(v^\star, \tau^\star)$, the timestep $\tau^\star$ has just been added to $\mathcal{R}(u)$ (and to $\mathsf{Awake}(u)$) in the call to FullUpdate$(u, \tau^\star)$ or to SimpleUpdate$(u, \tau^\star)$, and cannot yet be removed from $\mathsf{Awake}(u)$. So we start with $\tau_u = \tau^\star$. For a contradiction, if we remove $\tau^\star$ from $\mathsf{Awake}(u)$ in line (3.17), then all the constraints in $\mathfrak{L}^u(\tau^\star)$ must have become depleted. For each such constraint $C \in \mathfrak{L}^u(\tau^\star)$, the contributions to the RHS in (I3) during this procedure come only from the newly-added constraints $C(v^\star, \sigma, \tau^\star) \in \mathfrak{L}^{v^\star}(\tau^\star)$. So if all constraints in $\mathfrak{L}^u(\tau^\star)$ become depleted, the total dual objective raised during this procedure is at least

$$\sum_{C \in \mathfrak{L}^u(\tau^\star)} \sum_{C(v^\star,\sigma,\tau^\star) \in \mathfrak{L}^{v^\star}(\tau^\star) : C \in \sigma} b^{C(v^\star,\sigma,\tau^\star)} \, z_{C(v^\star,\sigma,\tau^\star)} \geq (1 + {}^1\!/_H) \sum_{C \in \mathfrak{L}^u(\tau^\star)} b^C \, z_C,$$

where we use that $b^{C(v^\star,\sigma,\tau^\star)} \geq b^C$ (because in (5.9), $b^{C_u} \geq 0$ by the induction hypothesis (Invariant (I2)) and $D(u, I_u) \geq 0$ by Corollary 5.2), and that each constraint in $\mathfrak{L}^u(\tau^\star)$ satisfies (I3) at equality. The induction hypothesis Invariant (I2) applied to $(u, \tau^\star)$ implies that $\sum_{C \in \mathfrak{L}^u(\tau^\star)} b^C \, z_C = \gamma$, so the RHS above is $(1 + {}^1\!/_H)\gamma$. So the total dual increase during FullUpdate$(v^\star, \tau^\star)$, which is at least the LHS above, is strictly more than γ, contradicting the stopping condition of FullUpdate$(v^\star, \tau^\star)$. ∎

Next, we prove the remainder of the inductive step, namely that Invariants (I2) and (I1) are satisfied with respect to $(v^\star, \tau^\star)$ as well.

Claim 5.6. (Inductive Step: Active Siblings Exist) *Consider the call* FullUpdate$(v^\star, \tau^\star)$, *and let u_0 be the principal child of $v^\star$ at this timestep. Suppose* $\mathsf{activesib}(u_0, \tau^\star) \neq \varnothing$. *Then the dual objective value corresponding to the constraints in $\mathfrak{L}^{v^\star}(\tau^\star)$ equals γ; i.e.,*

$$\sum_{C \in \mathfrak{L}^{v^\star}(\tau^\star)} z_C \, b^C = \gamma.$$

Moreover, the server mass entering $\mathbb{T}_{u_0}$ going to the requested source node in this call is at most

$$\frac{\gamma - \mathsf{loss}(u_0, \tau^\star)}{\lambda^{\mathsf{level}(u)}}.$$

Proof. Let $U' := \mathsf{activesib}(u_0, \tau^\star)$ be the non-principal children of $v^\star$ at timestep $\tau^\star$; let $U := \{u_0\} \cup U'$ as in FullUpdate. The identity of the timesteps τ_u and intervals I_u change over the course of the call, so we need notation to track them carefully. Let $I_u(s')$ be the set I_u when the timer value is s'; similarly, let $D_s(u, I_u(s'))$ be the value of $D(u, I_u(s'))$ when the timer value is s, and $y_s^{v^\star}(u, I_u(s'))$ is defined similarly.

For $u \in U'$, Corollary 5.2(ii,iii) implies that for any interval $I_u(s)$,

$$D_0(u, I_u(s)) \geq y_0^{v^\star}(u, I_u(s)). \tag{5.14}$$

Since the timestep $\tau^\star$ stays awake for the principal child u_0 (due to Claim 5.5), the interval $I_{u_0}(s)$ equals $(\tau^\star, \tau^\star]$, which is empty, for all values of the timer s.

The dual increase is *at most* γ due to the stopping criterion for FullUpdate, so we need to show this quantity reaches γ. Indeed, suppose we raise the timer from s to $s + ds$ when considering some constraint $C_s(v, \sigma, \tau)$—the subscript indicates the constraint considered at that value of timer s. The dual objective increases by $b^{C_s(v,\sigma,\tau)} \, ds$. We now use the definition of $b^{C_s(v,\sigma,\tau)}$ from (5.9), substitute $(n_v - \sum_{u \in U} n_u) \geq 1$, and use that all b^{C_u} terms in the summation are non-negative (by Invariant (I2)) to drop these terms. This gives the first inequality below (recall that $I_{u_0}(s)$ stays empty):

$$b^{C_s(v,\sigma,\tau)} \geq \sum_{u \in U'} D_s(u, I_u(s)) + \delta \geq \sum_{u \in U'} D_0(u, I_u(s)) + \delta \geq \sum_{u \in U'} y_0^{v^\star}(u, I_u(s)) + \delta. \tag{5.15}$$

The second inequality above uses that $D_s \geq D_0$ for non-principal children, and the third uses (5.14). Let

$$Y(s) := \sum_{\tau'} \sum_{u \in U'} \left(y_s^{v^\star}(u, \tau') - y_0^{v^\star}(u, \tau') \right)$$

to be the total increase in the $y^{v^\star}$ variables during FULLUPDATE$(v^\star, \tau^\star)$ until the timer reaches s. This is also the total amount of server transferred to the requested node due to the *local component* of transfer in line (3.16) until this moment.

SUBCLAIM 5.1. $Y(s) < \gamma$.

Proof. Suppose not, and let $s^\star$ be the smallest value of the timer such that $Y(s^\star) = \gamma$. Note that $Y(s)$ is a continuous non-decreasing function of s. For any $s \in [0, s^\star)$, we get $Y(s) < Y(0) + \gamma$, where $Y(0) = 0$ by definition. Since the intervals $I_u(s') \subseteq I_u(s)$ for $s' \leq s$, all the increases in the $y^{v^\star}$ variables during $[0, s]$ correspond to timesteps in $I_u(s)$. Thus for any $s < s^\star$,

$$Y(0) + \gamma > Y(s) \quad \Longrightarrow \quad \sum_{u \in U'} y_0^{v^\star}(u, I_u(s)) + \gamma > \sum_{u \in U'} y_s^{v^\star}(u, I_u(s)). \tag{5.16}$$

The dual increase during $[s, s + ds]$ is

$$\begin{aligned} b^{C_s(v,\sigma,\tau)}\, ds &\overset{\text{by } (5.15,5.16)}{>} \Big(\sum_{u \in U'} y_s^{v^\star}(u, I_u(s)) + \delta - \gamma\Big)\, ds \\ &= \Big(\lambda^h\, dY(s) - \frac{\gamma}{Mn} \sum_{u \in U'} |S_u|\, ds\Big) + (\delta - \gamma)\, ds > \lambda^h\, dY(s) \geq dY(s). \end{aligned}$$

The second line uses (a) the update rule in line (3.15) with $dY(s)$ denoting $Y(s + ds) - Y(s)$, (b) that $M \geq |S_u|$ and $|U'| \leq n$, so the second expression is bounded by γ, and (c) that $\delta > 2\gamma$. Integrating over $[0, s^\star]$, the total dual increase is strictly more than $Y(s^\star) = \gamma$, which contradicts the stopping condition of FULLUPDATE. ☐

Combining Subclaim 5.1 (and specifically its implication (5.16)) with (5.14) implies that for all values s of the timer:

$$\sum_{u \in U'} y_s^{v^\star}(u, I_u(s)) < \sum_{u \in U'} D_0(u, I_u(s)) + \gamma. \tag{5.17}$$

Therefore, the increase in dual objective during $[s, s + ds]$ is at least

$$\begin{aligned} b^{C_s(v,\sigma,\tau)}\, ds &\overset{(5.9)}{\geq} \Big(\sum_{u \in U'} \Big(D_0(u, I_u(s)) + b^{C_{u,s}}\Big) + \delta + b^{C_{u_0}}\Big)\, ds \\ &\overset{(5.17)}{>} \Big(\sum_{u \in U'} \Big(y_s^{v^\star}(u, I_u(s)) + b^{C_{u,s}}\Big) + (\delta - \gamma) + b^{C_{u_0}}\Big)\, ds \\ &\geq \sum_{u \in U'} \Big(\lambda^h\, dy_s^{v^\star}(u, I_u(s)) - \frac{\gamma}{Mn}|S_u|\, ds + b^{C_{u,s}}\, ds\Big) + \gamma\, ds + b^{C_{u_0}}\, ds \\ &\geq \sum_{u \in U'} \Big(\lambda^h\, dy_s^{v^\star}(u, I_u(s)) + b^{C_{u,s}}\Big)\, ds + b^{C_{u_0}}\, ds \\ &= \lambda^h[\text{amount of server transferred in } [s, s + ds]] + b^{C_{u_0}}\, ds \end{aligned}$$

Here $C_{u,s}$ is the constraint corresponding to $u \in U'$ when the timer equals s. The third inequality above follows from the update rule in line (3.15), and that $\delta \geq 2\gamma$. The last equality follows from line (3.16). Integrating over the entire range of the timer s, we see that the total dual objective increase is at least λ^h[total server transfer] + $\mathsf{loss}(u_0, \tau^\star)$. Since the total dual increase is at most γ, the total server transfer is at most $\frac{\gamma - \mathsf{loss}(u_0,\tau^\star)}{\lambda^h}$. This proves the second part of Claim 5.6.

We now prove that the FULLUPDATE process does not stop until the dual increase is γ. For each $u \in U'$, the subtree $\mathbb{T}_u$ contains at least one active leaf and hence at least δ servers when FULLUPDATE is called. Since the total server transfer is at most $\gamma \ll \delta$, we do not run out of servers. It follows that until the dual objective reaches γ, we keep raising $y_s^{v^\star}(u, I_u(s))$ for some non-empty interval $I_u(s)$ for each $u \in U'$, and this also raises the dual objective as above. ☐

It remains to consider the general case when $\mathsf{activesib}(u_0, \tau^\star)$ may be empty.

CLAIM 5.7. (INDUCTIVE STEP: GENERAL CASE) *At the end of any call* FULLUPDATE$(v^\star, \tau^\star)$*, the total dual objective raised during the call equals* γ.

Proof. If $\mathsf{activesib}(u_0, \tau^\star)$ is non-empty, this follows from Claim 5.6. So assume that $\mathsf{activesib}(u_0, \tau^\star)$ is empty. In this case, there are no $y^v(u,t)$ variables to raise because the interval I_{u_0} is empty. As we raise s, we also raise $z_{C(v^\star,\sigma,\tau^\star)}$ in line (3.13). Since we do not make all the constraints in $\mathfrak{L}^{u_0}(\tau^\star)$ depleted (Claim 5.7), the total dual increase must reach γ, because $b^{C(v^\star,\sigma,\tau^\star)} > 0$ by Corollary 5.2. ∎

This completes the proof of the induction hypothesis for the pair $(v^\star, \tau^\star)$.

We now give a simple upper bound on the parameter M.

COROLLARY 5.3. (BOUND ON M) *For node u and timestep τ, let $\tau_u := \mathsf{prev}(u, \tau)$. There are at most $\frac{5H\lambda^H(T+k)}{4\gamma}$ timesteps in $(\tau_u, \tau] \cap \mathcal{R}^{ns}(u)$. So we can set M to $\frac{5H\lambda^H(T+k)}{4\gamma} + 1$.*

Proof. Let $I := (\tau_u, \tau]$. By the choice of τ_u, none of the timesteps in $S := I \cap \mathcal{R}^{ns}(u)$ belongs to $\mathsf{Awake}(u)$. The proof of Corollary 5.2, and specifically (5.13), shows that $g^{\mathsf{inh}}(u, I) - r(u, I) \geq \frac{4|S|\gamma}{5H\lambda^h}$. This difference denotes the net server mass leaving the subtree $\mathbb{T}_u$ during I using the edge $(u, p(u))$. We claim that quantity cannot be more than $T + k$. Indeed, at the beginning of I, there are at most k servers in the sub-tree $\mathbb{T}_u$ and at each request time, at most one unit of server mass can enter this sub-tree through direct transfer of server mass from a source to the corresponding destination of a request. So $|S| \leq \frac{5H\lambda^H(T+k)}{4\gamma}$. Since the set $|S_u|$ defined in line 3.14 in FULLUPDATE is at most $|S| + 1$ (because of the first timestep of I_u), the desired result follows. ∎

5.3 Approximate Dual Feasibility For $\beta \geq 1$, a dual solution z is *β-feasible* if z/β satisfies the dual constraints. We now show that the dual variables raised during the calls to FULLUPDATE(v, τ) for various timesteps τ remain β-feasible for $\beta = O(\log \frac{nMk}{\gamma})$. First we show Invariant (I1), and also give bounds on variables $y^v(u,t)$.

CLAIM 5.8. (PROOF OF INVARIANT (I1)) *For any timestep τ and the requested source leaf $v = \mathbb{s}_{\lfloor \tau \rfloor}$, the server mass $k_{v,\tau}$ lies in the interval $(-1 - 4\delta n, -2\delta n]$. For any other source leaf node v', $k_{v',\tau}$ lies in the range $(-4\delta n, -2\delta n]$. Further, the server mass at each sink leaf is at least $\delta/2$.*

Proof. The proof is by induction on timestep τ. Let q denote $\lfloor \tau \rfloor$ and suppose the invariant holds for all timesteps before q. Before the request q arrives, the source node $\mathbb{s}_q$ has at least $-4\delta n$ servers (by induction hypothesis). Since we immediately move 1 unit of server mass out of this leaf node, it has at least $-1 - 4\delta n$ servers at time $\lfloor \tau \rfloor$. During $[q, q+1)$, we shall only move server mass into $\mathbb{s}_q$. Therefore, the server mass at $\mathbb{s}_q$ remains at least $-1 - 4\delta n$. We can show the upper bound similarly. Just before time q, the server mass is at most $-2\delta n$, and so it becomes at most $-1 - 2\delta n < -4\delta n$ after the movement of 1 unit of server mass out of $\mathbb{s}_q$. Now we keep transferring server mass in units of at most $2\gamma \leq 2\delta n$ till it exceeds $-4\delta n$. Therefore the final server mass at $\mathbb{s}_q$ will be at most $-2\delta n$ (and at least $-4\delta n$). This proves the desired result about source nodes.

The statement about sink nodes follows easily from the fact that we move server mass out of a sink node only when it has at least δ amount of server mass. Further at most $2\gamma \leq \delta/2$ server mass moves out of it during one call to FULLUPDATE. Therefore, the server mass at a sink leaf remains at least $\delta/2$. ∎

CLAIM 5.9. (BOUND ON y^v VALUES) *For any vertex v, any child u of v, and timestep τ, the variable $y^v(u,\tau) \leq T$.*

Proof. In each step, any variable $y^v(u,\tau)$ can increase by at most one, because any increase in this variable is accompanied by a corresponding transfer of server mass to the source node. This follows from that fact that server mass is being transferred at rate at least $\frac{dy^v(u,\tau)}{dt}$ and the total server mass transferred does not exceed one. ∎

CLAIM 5.10. *Let t be any timestep in $\mathcal{R}(u)$, and v be the parent of u. Define $\mathbb{t}_1$ to be the last timestep in $\mathcal{R}(u) \cap [0, t]$, and $\mathbb{t}_2$ to be the next timestep, i.e., $\mathbb{t}_1 + \eta$. Let C be a constraint in $\mathfrak{L}^v$ containing the variable $y^v(u,t)$ on the LHS. Then C contains at least one of $y^v(u,\mathbb{t}_1)$ and $y^v(u,\mathbb{t}_2)$. Moreover, whenever we raise $z(C)$ in line (3.13) of the* FULLUPDATE *procedure, we also raise either $y^v(u,\mathbb{t}_1)$ or $y^v(u,\mathbb{t}_2)$ according to line (3.15).*

Proof. Suppose $y^v(u,t)$ appears in a constraint $\mathfrak{L}^v(\tau)$. Define $I_u = (\tau_u, \tau]$ as in line (3.7). It follows that $t \in I_u$, and so $\tau_u < t$. Therefore, $\tau_u \in \mathcal{R}(u) \cap [0,t]$, so either $\mathbb{t}_1 > \tau_u$ and hence belongs to I_u, or else $\mathbb{t}_1 = \tau_u$ in which case $\mathbb{t}_2 \in I_u$. It follows that the index set S_u contains either $\mathbb{t}_1$ or $\mathbb{t}_2$. This implies the second statement in the claim. ∎

We now show the approximate dual feasibility. Recall that the constraints added to $\mathfrak{L}^v(\tau)$ are of the form $C(v,\sigma,\tau)$ given in (5.9), and we raise the corresponding dual variable $z_{C(v,\sigma,\tau)}$ only during the procedure FullUpdate(v,τ) and never again.

Lemma 5.5. (Approximate Dual Feasibility) *For a node v at height $h+1$, the dual variables z_C are β_h-feasible for the dual program $\mathfrak{D}^v$, where $\beta_h = (1+{}^1\!/_H)^h\, O(\ln n + \ln M + \ln(k/\gamma))$.*

Proof. We prove the claim by induction on the height of v. For a leaf node, this follows vacuously, since the primal/dual programs are empty. Suppose the claim is true for all nodes of height at most h. For a node v at height $h+1 > 0$ with children χ_v, the variables in $\mathfrak{L}^v$ are of two types: (i) $y^v(u,t)$ for some timestep t and child $u \in \chi_v$, and (ii) $y^v(u',t)$ for some timestep t and non-child descendant $u' \in \mathbb{T}_v \setminus \chi_v$. We consider these cases separately:

I. Suppose the dual constraint corresponds to variable $y^v(u,t)$ for some child $u \in \chi_v$. Let $\mathfrak{L}'$ be the set of constraints in $\mathfrak{L}^v$ containing $y^v(u,t)$ on the LHS. The dual constraint is:

$$\sum_{C \in \mathfrak{L}'} z_C \le c_u = \lambda^h. \tag{5.18}$$

Let $\mathbb{t}_1, \mathbb{t}_2$ be as in the statement of Claim 5.10. When we raise z_C for a constraint $C \in \mathfrak{L}'$ in line (3.13) at unit rate, we raise either $y^v(u,\tau_1)$ or $y^v(u,\mathbb{t}_2)$ at the rate given by line (3.15). Therefore, if we raise the LHS of the dual constraint (5.18) for a total of Γ units of the timer, we would have raised one of the two variables, say $y^v(u,\tau_1)$, for at least $\Gamma/2$ units of the timer. Therefore, the value of $y^v(u,\tau_1)$ variable due to this exponential update is at least

$$\frac{\gamma}{Mn}(e^{\Gamma/2\lambda^h} - 1).$$

By Claim 5.9, this is at most T, so we get

$$\Gamma = \lambda^h \cdot O(\ln n + \ln M + \ln(k/\gamma)) = \beta_0 c_u,$$

hence showing that (5.18) is satisfied up to β_0 factor (right now, M and T are same).

II. Suppose the dual constraint corresponds to some variable $y^v(u',\tau)$ with $u' \in \mathbb{T}_u$, and $u \in \chi_v$. Suppose u' is a node at height $h' < h$. Now let $\mathfrak{L}'$ be the constraints in $\mathfrak{L}^u$ (the LP for the child u) which contain $y^u(u',\tau)$. By the induction hypothesis:

$$\sum_{C \in \mathfrak{L}'} z_C \le \beta_{h-1}\, c_{u'}. \tag{5.19}$$

Let $\mathfrak{L}''$ denote the set of constraints in $\mathfrak{L}^v$ (the LP for the parent v) which contain $y^v(u',\tau)$. Each constraint $C(v,\sigma,\tau)$ in this set $\mathfrak{L}''$ has the coordinate σ_u corresponding to the child u being a constraint in $\mathfrak{L}'$, which implies:

$$\sum_{C(v,\sigma,\tau) \in \mathfrak{L}''} z_{C(v,\sigma,\tau)} = \sum_{C \in \mathfrak{L}'} \sum_{C(v,\sigma,\tau) \in \mathfrak{L}'' : \sigma_u = C} z_{C(v,\sigma,\tau)} \le (1 + {}^1\!/_H) \sum_{C \in \mathfrak{L}'} z_C, \tag{5.20}$$

where the last inequality uses Invariant (I1). Now the induction hypothesis (5.19) and the fact that $\beta_h = (1+{}^1\!/_H)\,\beta_{h-1}$ completes the proof.

∎

Lemma 5.5 means that the dual solution for $\mathfrak{L}^{\mathrm{r}}$ is β_H-feasible, where $\beta_H = O(\ln \frac{nMk}{\gamma})$. This proves Lemma 5.2 and completes the proof of our fractional k-server algorithm.

5.4 Removing dependence of the competitive ratio on T In this section, we show how to improve the competitive ratio bound by removing dependence on the length of the time horizon, namely T. For a vertex v, let $\mathsf{IN}_T(v)$ be the set of requests $(\mathbb{s}_q, \mathbb{t}_q)$ among the first T requests for which $\mathbb{s}_q \notin \mathbb{T}_v, \mathbb{t}_q \in \mathbb{T}_v$. Similarly, let $\mathsf{OUT}_T(v)$ denote the set of requests among the first T requests for which $\mathbb{s}_q \in \mathbb{T}_v$ but $\mathbb{t}_q \notin \mathbb{T}_v$. Finally, define $\mathsf{NET}_T(v) := |\mathsf{IN}_T(v)| - |\mathsf{OUT}_T(v)|$, and let NET_T denote $\max_v \mathsf{NET}_T(v)$.

Claim 5.11. *The cost of the optimal solution for the first T requests is at least $\frac{\mathsf{NET}_T}{k} - 1$.*

Proof. Consider the optimal solution. Let $\mathsf{IN}_{i,T}(v)$ be the subset of requests in $\mathsf{IN}_T(v)$ that are handled by server i. Define $\mathsf{OUT}_{i,T}(v)$ and $\mathsf{NET}_{i,T}(v)$ analogously. Let v be the vertex for which $\mathsf{NET}_T = \mathsf{NET}_T(v)$. There must be a server i for which $\mathsf{NET}_{i,T}(v) \geq {}^{\mathsf{NET}_T(v)}/_k$. Consider the sequence $\sigma_{i,v}$ of requests (ordered by arrival time) in $\mathsf{IN}_{i,T}(v) \cup \mathsf{OUT}_{i,T}(v)$. Consider two consecutive requests q_1, q_2 in this ordering and suppose both of these belong to the set $\mathsf{IN}_{i,T}(v)$. After reaching $\mathbb{t}_{q_1}$ (inside $\mathbb{T}_v$), the server can reach $\mathbb{s}_{q_2}$ only by traversing the edge $(v, p(v))$. Indeed there is no request in (q_1, q_2) that belongs to $\mathsf{OUT}_{i,T}(v)$. Since the sequence $\sigma_{i,v}$ has at least $\mathsf{NET}_{i,T}(v) - 1$ pairs of consecutive requests from $\mathsf{IN}_{i,T}(v)$, server i must traverse the edge $(v, p(v))$ at least $\mathsf{NET}_{i,T}(v) - 1$ times. Since the length of this edge is at least 1, the desired result follows. ∎

The following is a more precise restatement of Corollary 5.3. The proof of this result follows along the same lines as that of Corollary 5.3 (by observing that the parameter T can be replaced by $\mathsf{NET}_T(u)$).

Claim 5.12. (Bound on M) *For node u and timestep τ, let $\tau_u := \mathsf{prev}(u, \tau)$. There are at most $\frac{5H\lambda^H(\mathsf{NET}_T(u)+k)}{4\gamma}$ timesteps in $(\tau_u, \tau] \cap \mathcal{R}^{ns}(u)$. So we can set M to $\frac{5H\lambda^H(\mathsf{NET}_T(v)+k)}{4\gamma} + 1$.*

The online algorithm. Motivated by the above observations, our revised online algorithm is as follows. We divide the input sequence into phases. Whenever a new phase begins, we restart Algorithm 1 on the subsequent input, i.e., we ignore the solution maintained so far, bring all the servers to the starting configuration (as explained in Section 3.4) and run Algorithm 1. A phase ends when NET_T equals $ck(n+k)\Delta$, where c is a large enough constant and T is the number of requests served in the current phase (i.e., NET_T is calculated with respect to the current phase only). This completes the description of the online algorithm.

Analysis. We now analyze the competitive ratio of this algorithm. Suppose we run the algorithm for P phases. Let $\mathcal{I}$ denote the overall sequence of requests and $\mathcal{I}^p$ denote the requests arriving in phase p. Suppose phase p handles T_p requests. For a phase p, let $\mathsf{NET}^p_{T_p}$ denote the value of NET_{T_p} while considering the input $\mathcal{I}^p$ only. For a phase $p < P$, we know that $\mathsf{NET}^p_{T_p} = ck(n+k)\Delta$. Let opt_p denote the offline optimum for $\mathcal{I}^p$ along with the $n+1$ additional requests given in Section 3.4 when the starting configuration is the one in Section 3.4. Let opt denote the offline optimum for the overall input $\mathcal{I}$.

Claim 5.13. $\sum_{p=1}^{P} \mathsf{opt}_p \leq \mathsf{opt} + O((n+k) \cdot P\Delta) \leq 2\ \mathsf{opt} + O((n+k) \cdot \Delta)$.

Proof. Consider the offline optimal algorithm $\mathcal{O}$ for $\mathcal{I}$ of cost opt. Let σ_p be the configuration of servers in $\mathcal{O}$ when the input $\mathcal{I}^p$ begins. From $\mathcal{O}$, we can obtain an algorithm $\mathcal{O}^p$ for $\mathcal{I}^p$ as follows: we first serve the additional $n+1$ requests given in Section 3.4, then modify the server configuration to σ_p, and finally run $\mathcal{O}$ for the entire sequence $\mathcal{I}^p$. The cost of serving the $n+1$ additional requests is at most $O(n\Delta)$ and that of modifying the configuration to σ_p is $O(k\Delta)$. Therefore, opt_p is at most $O((n+k)\Delta)$ plus the cost incurred by $\mathcal{O}$ during $\mathcal{I}^p$. The first inequality in the claim follows from this observation.

It remains to show the second inequality. If $P = 1$, then there is nothing to show. So assume $P > 1$. For a phase $p < P$, we know that $\mathsf{NET}^p_{T_p} = ck(n+k)\Delta$. Claim 5.11 now implies that $\mathsf{opt}_p \geq c(n+k)\Delta - 1$. Thus, $\sum_{p=1}^{P} \mathsf{opt}_p \geq ({}^{c(n+k)}/_2) \cdot P\Delta$. The desired inequality now follows from this fact and the first inequality in the claim. ∎

Claim 5.14. *For a phase $p < P$, the cost incurred by the online algorithm is $O(H \log \frac{nMk}{\gamma})\mathsf{opt}_p$. For the last phase P, the cost incurred by the online algorithm is $O(H \log \frac{nMk}{\gamma})\mathsf{opt}_P + O(k\Delta)$.*

Proof. We know from the proof of Theorem 5.1 that the cost incurred by the online algorithm in a particular phase p is at most $O(\beta H)\mathsf{opt}_p + O(k\Delta)$, where $\beta = O(\log \frac{nMk}{\gamma})$. The additive term comes because of the cost of setting up the starting configuration. As argued in the proof of Claim 5.13, $\mathsf{opt}_p = \Omega((n+k)\Delta)$ if $p < P$. The desired result now follows from this observation. ∎

Using Claim 5.14, we get the following bound on the cost of the algorithm:

$$O\left(H \log \frac{nMk}{\gamma}\right) \sum_{p=1}^{P} \mathsf{opt}_p + O(k\Delta) \leq^{(\text{Claim 5.13})} O\left(H \log \frac{nMk}{\gamma}\right) \mathsf{opt} + O\left(H \log \frac{nMk}{\gamma}\right) \cdot (n+k)\Delta + O(k\Delta).$$

For each phase, Claim 5.12 (along with the fact that γ is $O(1/n^4)$) implies that the parameter M is at most $O(H\lambda^H k\Delta n^5) = O(\lambda^H k \Delta^2 n^6)$ since $H \leq n\Delta$. Using this value of M in the above bound on the cost of the algorithm, we get the following theorem:

Theorem 5.2. (Competitiveness for k-Taxi) *Given any instance of the k-Taxi problem on a λ-HST with height at most $\lambda/10$, the online algorithm ensures that each request location $\mathbb{s}_q$ is saturated at some timestep in $[q, q+1)$. The total cost of (fractional) server movement is $O(H^2 \log(nk\Delta))$ times the cost of the optimal solution (plus a fixed additive term $O\left(H^2 \log \frac{nk\Delta}{\gamma}\right) \cdot (n+k)\Delta$).*

6 The Rounding Algorithm

We now give the rounding algorithm for k-Taxi: while we cannot directly use the rounding algorithm of [BBMN15] and [BCL+18] because of the free movement of the server from $\mathbb{s}_q$ to $\mathbb{t}_q$, we show how to use the main ingredients from those rounding algorithms can still be used in a nearly black-box fashion.

6.1 Abstracting the Fractional Algorithm Define $f_q : L_{\mathbb{r}} \to \mathbb{R}$ to be fractional server assignments maintained by the above algorithm at the last timestep $q + 1 - \eta$ that corresponds to request q, it satisfies the following properties:

i. $f_q(\mathbb{t}_q) \geq 1$,
ii. $f_q(v) \geq -4\delta n$ for all source nodes, and
iii. $\sum_v f_q(v) = k$.

Moreover, we can abstract the process above as follows:

1. At time q, we start with the previous server locations f_{q-1}, and receive the source-sink pair $\mathbb{s}_q$-$\mathbb{t}_q$.
2. We move to an intermediate assignment $f_{q-1/2}$ defined by $f_{q-1/2}(\mathbb{s}_q) = f_q(\mathbb{s}_q) + 1$, $f_{q-1/2}(\mathbb{t}_q) = f_q(\mathbb{t}_q) - 1$, and $f_{q-1/2}(v) = f_q(v)$ for all other nodes v.
3. Finally, we move to the assignment f_q that fractionally satisfies request r_q by moving one unit of server mass from $\mathbb{s}_q$ to $\mathbb{t}_q$ at zero cost.

Given the HST T, define the earth-mover distance between f and f' to be

$$\text{EMD}(f, f') = \sum_{v \in V} c_v |f(v) - f'(v)|.$$

The cost incurred by the fractional algorithm is exactly $\sum_q \text{EMD}(f_q, f_{q+1/2})$, since the movement of the server from $f_{q+1/2}$ to f_{q+1} is free.

6.2 Reducing to the Rounding for k-Server For the rounding algorithm, we use two subroutines in an almost black-box fashion. The first is a result of Bubeck et al. [BCL+18, Lemma 3.4], which we paraphrase here:

Lemma 6.1. (Correcting Server Mass) *For every $0 \leq \zeta < 1$, there exists a map σ that takes any fractional server assignment f using $k+\zeta$ servers and outputs a new fractional server assignment $\sigma(f)$ using only k servers, such that $\sigma(f)(v) \in [\lfloor f(v) \rfloor, f(v)]$. Moreover, for any f, f',*

$$\text{EMD}(\sigma(f), \sigma(f')) \leq 1/(1-\zeta) \cdot \text{EMD}(f, f').$$

The second rounding result is by Bansal et al. [BBMN15, Proof of Theorem 5.2]. Define a randomized k-server state S (i.e., a distribution over integer k-server solutions) to be *consistent* for a fractional assignment f if $\Pr_S[\ell \in S] = f(\ell)$ for any leaf $\ell \in L_{\mathbb{r}}$, i.e., the marginals are correct at the leaves. Define it to be *balanced* for f if for each node v, and each of the deterministic configurations C in the support of S, the number of servers in configuration C that belong to the subtree T_v is either $\lfloor \sum_{\ell \in L_v} f(\ell) \rfloor$ or $\lceil \sum_{\ell \in L_v} f(\ell) \rceil$.

Lemma 6.2. (One-Step Rounding) *Consider a λ-HST with $\lambda > 5$. Given any fractional k-server assignments f, f' and any randomized k-server state S that is consistent and balanced for f, there exists a randomized k-server state S that is consistent and balanced for f' such that the expected movement cost from S to S' is at most a constant times* $\mathrm{EMD}(f, f')$.

Our rounding procedure does the following:

1. Define $\zeta := 4\delta n^2$; our choice of δ ensures that $\zeta \ll 1/2$. For each q, apply Lemma 6.1 to the fractional assignments $f_{q-1} + 4\delta n$ and $f_{q-1/2} + 4\delta n$, each of which has $k + \zeta$ fractional servers, to get new assignments $\hat{f}_{q-1}, \hat{f}_{q-1/2}$ such that
$$\mathrm{EMD}(\hat{f}_{q-1}, \hat{f}_{q-1/2}) \le O(1) \cdot \mathrm{EMD}(f_{q-1}, f_{q-1/2}).$$
Since $f_{q-1/2}(\mathbb{s}_q) + 4\delta n \ge 1$, Lemma 6.1 ensures that $\hat{f}_{q-1/2}(\mathbb{s}_q) \ge 1$ as well.

2. Given a consistent and balanced randomized k-server state S_{q-1} for $\hat{f}_{q-1}$, use Lemma 6.2 to obtain the (consistent and balanced) randomized state $S_{q-1/2}$ for $\hat{f}_{q-1/2}$. The balancedness ensures that each configuration C in the support of $S_{q-1/2}$ contains a server at $\mathbb{s}_q$. Now construct S_q by moving one server from $\mathbb{s}_q$ to $\mathbb{t}_q$ in each of these configurations in $S_{q-1/2}$. By construction S_q is consistent and balanced for $\hat{f}_q$. We now proceed by induction.

Since $\zeta \ll 1$ and $\lambda > 5$ in our constructions, we get the main result of this section:

Theorem 6.1. (Rounding) *Consider a λ-HST with $\lambda > 5$. The fractional server assignments produced by the algorithms in the previous section can be rounded online to get a randomized k-taxi algorithm, whose expected cost is at most a constant factor of the fractional cost.*

7 Concluding Remarks

In this work, we give a covering LP formulation for the k-Taxi problem for HSTs (and thence for general graphs), and use it to obtain an $O(\mathrm{polylog}(nk\Delta))$-competitive randomized algorithm; this is the first such algorithm for the problem. The main conceptual contribution is the isolation of a subset of covering constraints implied by the natural min-cost flow relaxation for k-Taxi, which are rich enough to effectively bound the optimal cost, yet malleable enough to allow us to account for the movement within each subtree using local LPs, using the framework of [GKP21].

Several open questions remain: (i) can we remove the dependence on Δ? (ii) can we extend the framework of [GKP21] to broader classes of metrical task/service systems, and its extension here to other metrical service systems with transformations? (iii) can we extend our results to k-Taxi with time windows?

References

[ACN00] Dimitris Achlioptas, Marek Chrobak, and John Noga. Competitive analysis of randomized paging algorithms. *Theor. Comput. Sci.*, 234(1-2):203–218, 2000.

[BBCS21] Sébastien Bubeck, Niv Buchbinder, Christian Coester, and Mark Sellke. Metrical service systems with transformations. In James R. Lee, editor, *12th Innovations in Theoretical Computer Science Conference, ITCS 2021, January 6-8, 2021, Virtual Conference*, volume 185 of *LIPIcs*, pages 21:1–21:20. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021.

[BBMN15] Nikhil Bansal, Niv Buchbinder, Aleksander Madry, and Joseph Naor. A polylogarithmic-competitive algorithm for the k-server problem. *J. ACM*, 62(5):40, 2015.

[BBN10] Nikhil Bansal, Niv Buchbinder, and Joseph Naor. Metrical task systems and the k-server problem on HSTs. In *ICALP*, volume 6198 of *Lecture Notes in Computer Science*, pages 287–298. Springer, 2010.

[BBN12a] Nikhil Bansal, Niv Buchbinder, and Joseph Naor. A primal-dual randomized algorithm for weighted paging. *J. ACM*, 59(4):19:1–19:24, 2012.

[BBN12b] Nikhil Bansal, Niv Buchbinder, and Joseph Naor. Randomized competitive algorithms for generalized caching. *SIAM J. Comput.*, 41(2):391–414, 2012.

[BCL$^+$18] Sébastien Bubeck, Michael B. Cohen, Yin Tat Lee, James R. Lee, and Aleksander Madry. 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*, pages 3–16. ACM, 2018.

[BCN21] Niv Buchbinder, Christian Coester, and Joseph (Seffi) Naor. Online k-taxi via double coverage and time-reverse primal-dual. In Mohit Singh and David P. Williamson, editors, *Integer Programming and Combinatorial Optimization - 22nd International Conference, IPCO 2021, Atlanta, GA, USA, May 19-21, 2021, Proceedings*, volume 12707 of *Lecture Notes in Computer Science*, pages 15–29. Springer, 2021.

[BCR23] Sébastien Bubeck, Christian Coester, and Yuval Rabani. The randomized k-server conjecture is false! 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 581–594. ACM, 2023.

[BG00] Yair Bartal and Eddie Grove. The harmonic k-server algorithm is competitive. *J. ACM*, 47(1):1–15, 2000.

[BGMN19] Niv Buchbinder, Anupam Gupta, Marco Molinaro, and Joseph (Seffi) Naor. k-servers with a smile: Online algorithms via projections. In *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2019, San Diego, California, USA, January 6-9, 2019*, pages 98–116. SIAM, 2019.

[BN09] Niv Buchbinder and Joseph (Seffi) Naor. Online primal-dual algorithms for covering and packing. *Math. Oper. Res.*, 34:270–286, May 2009.

[CK19] Christian Coester and Elias Koutsoupias. The online k-taxi problem. In Moses Charikar and Edith Cohen, editors, *Proceedings of the 51st Annual ACM SIGACT Symposium on Theory of Computing, STOC 2019, Phoenix, AZ, USA, June 23-26, 2019*, pages 1136–1147. ACM, 2019.

[CKPV91] Marek Chrobak, H Karloff, T Payne, and S Vishwnathan. New results on server problems. *SIAM Journal on Discrete Mathematics*, 4(2):172–181, 1991.

[CL91] Marek Chrobak and Lawrence L Larmore. An optimal on-line algorithm for k servers on trees. *SIAM Journal on Computing*, 20(1):144–148, 1991.

[CL06] Béla Csaba and Sachin Lodha. A randomized on-line algorithm for the k-server problem on a line. *Random Struct. Algorithms*, 29(1):82–104, 2006.

[CMP08] Aaron Cote, Adam Meyerson, and Laura J. Poplawski. Randomized k-server on hierarchical binary trees. In Cynthia Dwork, editor, *Proceedings of the 40th Annual ACM Symposium on Theory of Computing, Victoria, British Columbia, Canada, May 17-20, 2008*, pages 227–234. ACM, 2008.

[DEH$^+$17] Sina Dehghani, Soheil Ehsani, MohammadTaghi Hajiaghayi, Vahid Liaghat, and Saeed Seddighin. Stochastic k-server: How should uber work? In Ioannis Chatzigiannakis, Piotr Indyk, Fabian Kuhn, and Anca Muscholl, editors, *44th International Colloquium on Automata, Languages, and Programming, ICALP 2017, July 10-14, 2017, Warsaw, Poland*, volume 80 of *LIPIcs*, pages 126:1–126:14. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2017.

[FKL$^+$91] Amos Fiat, Richard M. Karp, Michael Luby, Lyle A. McGeoch, Daniel Dominic Sleator, and Neal E. Young. Competitive paging algorithms. *J. Algorithms*, 12(4):685–699, 1991.

[FRR90] Amos Fiat, Yuval Rabani, and Yiftach Ravid. Competitive k-server algorithms (extended abstract). In *31st Annual Symposium on Foundations of Computer Science, St. Louis, Missouri, USA, October 22-24, 1990, Volume II*, pages 454–463. IEEE Computer Society, 1990.

[FRR94] Amos Fiat, Yuval Rabani, and Yiftach Ravid. Competitive k-server algorithms. *J. Comput. Syst. Sci.*, 48(3):410–428, 1994.

[FRT04] Jittat Fakcharoenphol, Satish Rao, and Kunal Talwar. A tight bound on approximating arbitrary metrics by tree metrics. *J. Comput. Syst. Sci.*, 69(3):485–497, 2004.

[GKP21] 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.

[GN14] Anupam Gupta and Viswanath Nagarajan. Approximating sparse covering integer programs online. *Mathematics of Operations Research*, 39(4):998–1011, 2014.

[Gro91] Edward F. Grove. The harmonic online k-server algorithm is competitive. In Cris Koutsougeras and Jeffrey Scott Vitter, editors, *Proceedings of the 23rd Annual ACM Symposium on Theory of Computing, May 5-8, 1991, New Orleans, Louisiana, USA*, pages 260–266. ACM, 1991.

[KP95] Elias Koutsoupias and Christos H. Papadimitriou. On the k-server conjecture. *J. ACM*, 42(5):971–983, 1995.

[MS91] Lyle A. McGeoch and Daniel Dominic Sleator. A strongly competitive randomized paging algorithm. *Algorithmica*, 6(6):816–825, 1991.

[Sei01] Steven S. Seiden. A general decomposition theorem for the k-server problem. In Friedhelm Meyer auf der Heide, editor, *Algorithms - ESA 2001, 9th Annual European Symposium, Aarhus, Denmark, August 28-31, 2001, Proceedings*, volume 2161 of *Lecture Notes in Computer Science*, pages 86–97. Springer, 2001.