


Online Multi-Level Aggregation with Delays and Stochastic Arrivals

Mathieu Mari ✉ 

LIRMM, University of Montpellier, France

Michał Pawłowski ✉ 

University of Warsaw, Poland

IDEAS NCBR, Warsaw, Poland

Sapienza University of Rome, Italy

Runtian Ren ✉ 

IDEAS NCBR, Warsaw, Poland

University of Wrocław, Poland

Piotr Sankowski ✉ 

University of Warsaw, Poland

IDEAS NCBR, Poland

MIM Solutions, Warsaw, Poland

Abstract

This paper presents a new research direction for online Multi-Level Aggregation (MLA) with delays. Given an *edge-weighted rooted tree* T as input, a *sequence of requests* arriving at its vertices needs to be served in an online manner. A request r is characterized by two parameters: its *arrival time* $t(r) > 0$ and *location* $l(r)$ being a vertex in tree T . Once r arrives, we can either serve it immediately or postpone this action until any time $t > t(r)$. A request that has not been served at its arrival time is called pending up to the moment it gets served. We can serve several pending requests at the same time, paying a service cost equal to the weight of the subtree containing the locations of all the requests served and the root of T . Postponing the service of a request r to time $t > t(r)$ generates an additional delay cost of $t - t(r)$. The goal is to serve all requests in an online manner such that the total cost (i.e., the total sum of service and delay costs) is minimized. The MLA problem is a generalization of several well-studied problems, including the TCP Acknowledgment (trees of depth 1), Joint Replenishment (depth 2), and Multi-Level Message Aggregation (arbitrary depth). The current best algorithm achieves a competitive ratio of $O(d^2)$, where d denotes the depth of the tree.

Here, we consider a stochastic version of MLA where the requests follow a Poisson arrival process. We present a deterministic online algorithm that achieves a constant ratio of expectations, meaning that the ratio between the expected costs of the solution generated by our algorithm and the optimal offline solution is bounded by a constant. Our algorithm is obtained by carefully combining two strategies. In the first one, we plan periodic oblivious visits to the subset of frequent vertices, whereas, in the second one, we greedily serve the pending requests in the remaining vertices. This problem is complex enough to demonstrate a very rare phenomenon that “single-minded” or “sample-average” strategies are not enough in stochastic optimization.

2012 ACM Subject Classification Theory of computation → Online algorithms

Keywords and phrases online algorithms, online network design, stochastic model, Poisson arrivals

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2024.49

Related Version *Full Version*: <https://arxiv.org/abs/2404.09711> [57]

Funding This work is supported by ERC CoG grant TUGbOAT no 772346, NCN grant no 2020/37/B/ST6/04179, and NCN grant no 2022/45/B/ST6/00559.



© Mathieu Mari, Michał Pawłowski, Runtian Ren, and Piotr Sankowski;
licensed under Creative Commons License CC-BY 4.0

35th International Symposium on Algorithms and Computation (ISAAC 2024).

Editors: Julián Mestre and Anthony Wirth; Article No. 49; pp. 49:1–49:20

Leibniz International Proceedings in Informatics



LIPICs Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

1 Introduction

Imagine the manager of a factory in charge of delivering products from the factory to the stores' locations. Once some products are in shortage for a store, its owner informs the factory for replenishment. From the factory's perspective, each time a service is created to deliver the products, a truck has to travel from the factory to the requested stores' locations and then return to the factory. A cost proportional to the total traveling distance is paid for this service. For the purpose of saving delivery costs, it is beneficial to accumulate the replenishment requests from many stores and then deliver the ordered products altogether in one service. However, this accumulated delay in delivering products may leave the stores unsatisfied, and the complaints will negatively influence future contracts between the stores and the factory. Typically, for each request ordered from a store, the time gap between ordering the products and receiving the products is known as the delay cost. The goal of the factory manager is to plan the delivery service schedule in an online manner such that the total service cost and the total delay cost are minimized.

The above is an example of an online problem called Multi-level Aggregation (MLA) with linear delays. Formally, the input is an edge-weighted rooted tree T and a sequence of requests, with each request r specified by an arrival time $t(r)$ and a location at a particular vertex. Once a request r arrives, its service does not have to be processed immediately but can be delayed to any time $t \geq t(r)$ at a delay cost of $t - t(r)$. The benefit of delaying requests is that several requests can be served together to save some service costs. To serve any set of requests R at time t , a subtree T' containing the tree root and locations of all the requests in R needs to be bought at a service cost equal to the total weight of edges in T' . The goal is to serve all requests in an online manner such that the total cost (i.e., the total service cost plus the total delay cost) is minimized.

Due to many real-life applications ranging from logistics, supply chain management, and data transmission in sensor networks, the MLA problem has recently drawn considerable attention [22, 16, 27, 13]. Besides, two classic problems in this area, TCP-acknowledgment (also known as a lot-sizing problem) and Joint Replenishment (JRP), are special cases of MLA with tree depths of 1 and 2, respectively. They were also extensively studied [32, 45, 63, 1, 47, 28, 21, 3, 60, 20]. Particularly for MLA, the current best online algorithm achieves a competitive ratio of $O(d^2)$ [13], where d denotes the depth of the given tree.

However, it is often too pessimistic to assume no stochastic information on the input is available in practice – again, consider our delivery example. The factory knows all the historical orders and can estimate the request frequencies from the stores of all locations. It is reasonable to assume that the requests follow some stochastic distribution. Therefore, the following question is natural: *if the input follows some stochastic distribution, can we devise online algorithms for MLA with better performance guarantees?*

In this paper, we provide an affirmative answer to this question. We study a stochastic online version of MLA, assuming that the requests arrive following a Poisson arrival process. More precisely, the waiting time between any two consecutive requests arriving at the same vertex u follows an exponential distribution $\text{Exp}(\lambda(u))$ with parameter $\lambda(u)$. In this model, the goal is to minimize the expected cost produced by an algorithm ALG for a random input sequence generated in a long time interval $[0, \tau]$. To evaluate the performance of ALG on stochastic inputs, we use the *ratio of expectations* (RoE) by comparing the expected cost of ALG with the expected cost of the optimal offline solution OPT (see Definition 6).

Our contribution. We prove that the performance guarantee obtained in the Poisson arrival model is significantly better compared with the current best competitiveness obtained in the adversarial model. More specifically, we propose a non-trivial deterministic online algorithm that achieves a constant ratio of expectations.

► **Theorem 1.** *For MLA with linear delays in the Poisson arrival model, there exists a deterministic online algorithm that achieves a constant ratio of expectations.*

Our algorithm is obtained by synergistically merging two carefully crafted strategies. The first strategy incorporates periodic oblivious visits to a subset of frequently accessed vertices, while the second strategy employs a proactive, greedy approach to handle pending requests in the remaining vertices. The complexity of this problem unveils a rare phenomenon – the inadequacy of “single-minded” or “sample-average” strategies in stochastic optimization. In this paper, we not only address this challenge but also point to further complex problems (such as facility location with delays [13] or online service with delays [9]) that require a similar approach in stochastic environments.

Previous works. The MLA problem has been only studied in the adversarial model. Bienkowski et al. [16] introduced a general version of MLA, assuming that the cost of delaying a request r by a duration t is $f_r(t)$. Here, $f_r(\cdot)$ denotes the delay cost function of r , which needs to be non-decreasing and satisfy $f_r(0) = 0$. They proposed an $O(d^4 2^d)$ -competitive online algorithm for this general delay cost version problem, where d denotes the tree depth [16, Theorem 4.2]. Later, the competitive ratio is further improved to $O(d^2)$ by Azar and Touitou [13, Theorem IV.2] (for the general delay cost version). However, no matching lower bound has been found for the delay cost version of MLA – the current best lower bound on MLA (with delays) is 4 [16, Theorem 6.3], restricted to a path case with linear delays. Thus far, no previous work has studied MLA in the stochastic input model.

Organization. We give the notations and preliminaries in Section 2. As a warm-up, we study a special single-edge tree instance in Section 3. We show that there are two different situations, we call them *heavy* case and *light* case, and to achieve a constant ratio of expectations, the ideas for the two cases are different. In Section 4, we give an overview of our deterministic online algorithm (Theorem 1). This algorithm is the combination of two different strategies for two different types of instances.¹ In Section 5, we study the *heavy* instances as a generalization of heavy single-edge trees. In Section 6, we prove the main Theorem 1. In Section 7, we provide some extra related works in detail. We finish the paper by discussing some future directions in Section 8.

2 Notations and preliminaries

Weighted tree. Consider an edge-weighted tree T rooted at vertex $\gamma(T)$. We refer to its vertex set by $V(T)$ and its edge set by $E(T)$. When the context is clear, we denote the root vertex, vertex set, and edge set by γ , V , and E , respectively. We assume that each edge $e \in E$ has a positive weight w_e . For any vertex $u \in V$, except for the root vertex γ , we denote its *parent vertex* as $\text{par}(u) \in V$, and $e_u = (u, \text{par}(u))$ as the edge connecting u and its parent. We also define T_u as the subtree of T rooted at vertex u . In addition to the edge weights, we use the term *vertex weight* to refer to $w_u := w(e_u)$, where $u \in V$ and $u \neq \gamma$.

¹ Due to the space limits, all the results for the light instances are ignored here but can be found in the full version [57].

Given any two vertices $u, v \in V(T)$, we denote the path length from u to v in T by $d_T(u, v)$, i.e., it is the total weight of the edges along this path. Finally, we use $T[U]$ to denote the forest induced by vertices of $U \subseteq V(T)$ in T .

Problem description. An MLA *instance* is characterized by a tuple (T, σ) , where T is a weighted tree rooted at γ and σ is a sequence of *requests*. Each request r is described by a tuple $(t(r), l(r))$ where $t(r) \in \mathbb{R}^+$ denotes r 's *arrival time* and $l(r) \in V(T)$ denotes r 's *location*. Thus, denoting by m the number of requests, we can rewrite $\sigma := (r_1, \dots, r_m)$ with the requests sorted in increasing order of their arrival times, i.e., $t(r_1) \leq t(r_2) \leq \dots \leq t(r_m)$. Given a sequence of requests σ , a *service* $s = (t(s), R(s))$ is characterized by the *service time* $t(s)$ and the set of requests $R(s) \subseteq \sigma$ it serves. A *schedule* S for σ is a sequence of services. We call schedule S *valid* for σ if each request $r \in \sigma$ is assigned a service $s \in S$ that does not precede r 's arrival. In other words, a valid S for σ satisfies (i) $\forall s \in S \forall r \in R(s) t(r) \leq t(s)$; (ii) $\{R(s) : s \in S\}$ forms a partition of σ . Given any MLA instance (T, σ) , an MLA algorithm ALG needs to produce a valid schedule S to serve all the requests in σ . Particularly, for an online MLA algorithm ALG, at any time t , the decision to create a service to serve a set of pending request(s) cannot depend on the requests arriving after time t . For each request $r \in \sigma$, let $S(r)$ denote the service in S which serves r , i.e., for each $s \in S$, $S(r) = s$ if and only if $r \in R(s)$. Given a sequence of requests σ and a valid schedule S , the *delay cost* for a request $r \in \sigma$ is defined as $\text{delay}(r) := t(S(r)) - t(r)$. Using this notion, we define the *delay cost* for a service $s \in S$ and the *delay cost* for the schedule S as $\text{delay}(s) := \sum_{r \in R(s)} \text{delay}(r)$ and $\text{delay}(S) := \sum_{s \in S} \text{delay}(s)$. Besides, given any $r \in \sigma$, if it is pending at time t , let $\text{delay}(r, t) = t - t(r)$ denote its delay cost at this moment.

The *weight* (also called *service cost*) of a service $s \in S$, denoted by $\text{weight}(s, T)$, is defined as the weight of the minimal subtree of T that contains root γ and all locations of requests $R(s)$ served by s . The *weight* (or *service cost*) of a schedule S is defined as $\text{weight}(S, T) := \sum_{s \in S} \text{weight}(s, T)$. To compute the *cost* of a service s , we sum its delay cost and weight, i.e., $\text{cost}(s, T) := \text{delay}(s) + \text{weight}(s, T)$. Similarly, we define the *cost* (or *total cost*) of a schedule S for σ as $\text{cost}(S, T) := \text{delay}(S) + \text{weight}(S, T)$. When the context is clear, we simply write $\text{cost}(S) = \text{cost}(S, T)$. Moreover, given an MLA instance (T, σ) , let $\text{ALG}(\sigma)$ denote the schedule of algorithm ALG for σ and let $\text{OPT}(\sigma)$ denote the optimal schedule for σ with minimum total cost. Note that without loss of generality, we can assume that no request in σ arrives at the tree root γ since such a request can be served immediately at its arrival with zero cost.

Poisson arrival model. Instead of using an adversarial model, we assume that the requests arrive according to some stochastic process. A *stochastic instance* is characterized by a tuple (T, λ) , where T denotes an edge-weighted rooted tree, and $\lambda : V(T) \rightarrow \mathbb{R}_+$ is a function that assigns each vertex $u \in V(T)$ an *arrival rate* $\lambda(u) \geq 0$. With no loss we assume $\lambda(\gamma(T)) = 0$, i.e., no request arrives at the tree root. Formally, such a tuple defines the following process.

► **Definition 2** (Poisson arrival model). *Given any stochastic MLA instance (T, λ) and any $\tau > 0$, we say that a (random) requests sequence σ follows a Poisson arrival model over time interval $[0, \tau]$, if (i) for each vertex $u \in V(T)$ with $\lambda(u) > 0$ the waiting time between any two consecutive requests arriving at u follows an exponential distribution with parameter $\lambda(u)$;² (ii) variables representing waiting times are mutually independent; (iii) all the requests in σ arrive within time interval $[0, \tau]$. We denote this fact by writing $\sigma \sim (T, \lambda)^\tau$.*

² For the first request r arriving at u , the waiting time from 0 to $t(r)$ follows the distribution $\text{Exp}(\lambda(u))$. Similarly, for the last request r' arriving at u , denoting by $W_{r'} \sim \text{Exp}(\lambda(u))$ its waiting time, we require that $\tau - t(r') < W_{r'}$.

Given any subtree T' of T , we use both $\lambda|_{T'}$ and $\lambda|_{V(T')}$ to denote the arrival rates restricted to the vertices of T' . Similarly, given a random sequence of requests $\sigma \sim (T, \lambda)^\tau$, we use $\sigma|_{T'} \subseteq \sigma$ and $\sigma|_I \subseteq \sigma$ for $I \subseteq [0, \tau]$ to denote the sequences of all requests in σ that arrive inside the subtree T' and within the time interval I , respectively.

In the following, we introduce three more properties of the Poisson arrival model. To simplify their statements, we denote the random variable representing the number of requests in sequence $\sigma \sim (T, \lambda)^\tau$ by $N(\sigma)$. The first property describes the expected value of $N(\sigma)$ for a fixed time horizon τ . The second one describes our model's behavior under the assumption that we are given the value of $N(\sigma)$. Finally, the third one presents the value of the expected waiting time generated by all the requests arriving before a fixed time horizon. All the proofs can be found in [62] or the full version of this paper [57].

► **Proposition 3.** *Given any stochastic MLA instance (T, λ) and a random sequence of requests $\sigma \sim (T, \lambda)^\tau$, it holds that: (i) $N(\sigma) \sim \text{Pois}(\lambda(T) \cdot \tau)$; (ii) $\mathbb{E}[N(\sigma) \mid \sigma \sim (T, \lambda)^\tau] = \lambda(T) \cdot \tau$; (iii) if $\lambda(T) \cdot \tau \geq 1$, then $\mathbb{P}(N(\sigma) \geq \mathbb{E}[N(\sigma)]) \geq 1/2$.*

► **Proposition 4.** *Given n requests arriving during time interval $[0, \tau]$ according to Poisson arrival model, the n arrival times (in sequence) have the same distribution as the order statistics corresponding to n independent random variables uniformly distributed over $[0, \tau]$.*

► **Proposition 5.** *Given any stochastic MLA instance (T, λ) and a random sequence of requests $\sigma \sim (T, \lambda)^\tau$, the expected delay cost of all the requests in $\sigma \sim (T, \lambda)^\tau$, assuming that no service was issued before τ , is $\mathbb{E}[\sum_{i=1}^{N(\sigma)} (\tau - t(r_i))] = \mathbb{E}[N(\sigma)] \cdot \tau/2 = \lambda(T) \cdot \tau^2/2$.*

Benchmark description. To measure the performance of an online algorithm ALG in this stochastic version of MLA, we use the *ratio of expectations*. Let $\mathbb{E}[\text{cost}(\text{ALG}(\sigma), T)]$ denote the expected cost of the schedule ALG generates for a random sequence $\sigma \sim (T, \lambda)^\tau$.

► **Definition 6 (ratio of expectations).** *An online algorithm ALG has a ratio of expectations (RoE) $C \geq 1$ if $\lim_{\tau \rightarrow \infty} \frac{\mathbb{E}[\text{cost}(\text{ALG}(\sigma), T)]}{\mathbb{E}[\text{cost}(\text{OPT}(\sigma), T)]} \leq C$ for any stochastic MLA instance (T, λ) .*

3 Warm-up: single edge instances

We start by considering the case of a single-edge tree in the stochastic model. That is, we fix a tree T that consists of a single edge $e = (u, \gamma)$ of weight $w > 0$ and denote the arrival rate of u by $\lambda > 0$. In such a setting, the problem of finding the optimal schedule to serve the requests arriving at vertex u is known as *TCP acknowledgment*. It is worth mentioning that in the adversarial setting, a 2-competitive deterministic and a $(1 - 1/e)^{-1}$ -competitive randomized algorithms are known for this problem [32, 45].

Let us stress that the goal of this section is not to improve the best-known competitive ratio for a single-edge case but to illustrate the efficiency of two opposite strategies and introduce some important concepts of this paper. The first strategy, called the *instant strategy*, is to serve each request as soon as it arrives. Intuitively, this approach is efficient when the requests are not so frequent so that, on average, the cost of delaying a request to the arrival time of the next request is enough to compensate for the service cost. The second strategy, called the *periodic approach*, is meant to work in the opposite case where requests are frequent enough so that it is worth grouping several of them for the same service. In this way, the weight cost of a service can be shared between the requests served. Assuming that requests follow some stochastic assumptions, it makes sense to enforce that services are ordered at regular time steps, where the time between any two consecutive services is a fixed number p , which depends only on the instance's parameters.

There are two challenges here. First, when should we use each strategy? Second, what should be the value of p that optimizes the performance of the periodic strategy? For the first one, we show that it depends on the value of $\pi := w\lambda$ that we call the *heaviness* of the instance. Specifically, we show that if $\pi > 1$, i.e., the instance is *heavy*, and the periodic strategy is more efficient. On the other hand, if $\pi \leq 1$, the instance is *light*, and the instant strategy is essentially better. For the second one, we show that the right value for the period, up to a constant in the ratio of expectations, is $p = \sqrt{2w/\lambda}$. With no loss, in what follows, we assume that the time horizon τ is always a multiple of the period chosen, which simplifies the calculation and does not affect the ratio of expectations.

► **Lemma 7.** *Given a stochastic instance where the tree is a single edge of weight w and the leaf has an arrival rate $\lambda > 0$, let $\pi = w\lambda$ and let σ be a random sequence of requests of duration $\tau > 0$. Then,*

1. *the instant strategy on σ has an expected cost of $\tau\pi$;*
2. *the periodic strategy on σ , with period $p = \sqrt{2w/\lambda}$, has an expected cost of $\tau\sqrt{2\pi}$.*

Note that the instant strategy incurs an expected cost equal to the expected number of requests arriving within the time horizon τ multiplied by the cost of serving one. By Proposition 3, we have that on average $\lambda\tau$ requests arrive within the time interval $[0, \tau]$. Thus, since the cost of serving one equals w , the total expected cost is $\lambda\tau w = \tau\pi$. For the periodic strategy, we know that within each period $p = \sqrt{2w/\lambda}$, we generate the expected delay cost of $\frac{1}{2} \cdot \lambda p^2 = w$ (Proposition 5). The service cost we pay at the end of each period equals w as well. Thus, the total expected cost within $[0, \tau]$ is equal to $\frac{\tau}{p} \cdot 2w = \tau \cdot \sqrt{2\lambda w} = \tau\sqrt{2\pi}$, which ends the proof.

We now compare these expected costs with the expected cost of the optimal offline schedule. The bounds obtained imply that the instant strategy has constant RoE when $\pi \leq 1$, and the periodic strategy (with $p = \sqrt{2w/\lambda}$) has a constant RoE when $\pi > 1$.

► **Lemma 8.** *Given a stochastic instance where the tree is a single edge of weight w and the leaf has an arrival rate $\lambda > 0$, let $\pi = w\lambda$ and let σ be a random sequence of requests of duration $\tau > 0$. For the lower bounds on the optimal offline schedule, $\text{OPT}(\sigma)$, it holds that*

1. *if $\pi \leq 1$, it has an expected cost of at least $\frac{1-e^{-1}}{2}\tau\pi$;*
2. *if $\pi > 1$, it has an expected cost of at least $\frac{3}{16}\tau\sqrt{2\pi}$.*

Due to the space limit, we only provide a proof sketch of Lemma 8 as follows. The main idea is to partition the initial time horizon $[0, \tau]$ into a collection of shorter intervals $\{I_1, I_2, \dots, I_k\}$ of length p each, for some value p that will be defined later. Denoting by $\sigma_i := \sigma|_{I_i}$ for $i \in [k]$, we know that all σ_i are independent and follow the same Poisson arrival model $(T, \lambda)^p$. Let $D(\sigma_1)$ denote the total delay cost of σ_1 at time p when no services are issued during $[0, p]$. Note that OPT either serves some requests during $[0, p]$ and incurs the service cost of at least w or issues no services during $[0, p]$ and pays the delay cost of $D(\sigma_1)$. The total cost of OPT within $[0, p]$ is thus at least $\min(w, D(\sigma_1))$ and hence

$$\mathbb{E}[\text{cost}(\text{OPT}(\sigma))] \geq \frac{\tau}{p} \cdot \mathbb{E}[\min(w, D(\sigma_1)) \mid \sigma_1 \sim (T, \lambda)^p].$$

Define $U_j = p - t(r_j)$ for the j -th request r_j in σ_1 . If $\pi \leq 1$, we choose $p = \frac{1}{\lambda}$, for which $\mathbb{P}(N(\sigma_1) \geq 1) = 1 - e^{-1}$ and $\mathbb{E}[\min(w, U_1)] \geq \frac{w}{2}$. This implies that

$$\mathbb{E}[\text{cost}(\text{OPT}(\sigma))] \geq \frac{1-e^{-1}}{2} \cdot \tau \cdot \lambda w.$$

If $w\lambda > 1$, we set $p = \sqrt{2w/\lambda}$ and $n_0 = \lceil \lambda p \rceil$, for which $\mathbb{P}(N(\sigma) \geq n_0) \geq \frac{1}{2}$ and

$$\mathbb{E}[\min(w, \sum_{j=1}^{n_0} U_j)] \geq 1 - \frac{w}{2n_0 p} \geq \frac{3}{4}.$$

This implies

$$\mathbb{E}[\text{cost}(\text{OPT}(\sigma))] \geq \frac{3}{16} \cdot \tau \cdot \sqrt{2w\lambda}.$$

See the appendix in the full version [57] for a detailed proof.

4 Overview

We now give an overview of the following sections. Inspired by the two strategies for the single edge instance, we define two types of stochastic instances: the *light* instances, for which the strategy of serving requests instantly achieves a constant RoE, and the *heavy* instances, for which the strategy of serving requests periodically achieves a constant RoE. Heavy and light instances are defined precisely below (Definitions 9 and 13) and generalize the notions of heavy and light single-edge trees studied in the previous section.

We define the light instances by extending the notion of *heaviness* for an arbitrary tree.

► **Definition 9.** A stochastic MLA instance (T, λ) is called *light* if $\pi(T, \lambda) \leq 1$, where $\pi(T, \lambda) := \sum_{u \in V(T)} \lambda(u) \cdot d(u, \gamma(T))$ is called the *heaviness* of the instance.

For a light instance, serving the requests immediately at their arrival time achieves a constant ratio of expectations. We refer to the schedule produced with this strategy (see Algorithm INSTANT in the full version [57]) on a sequence of requests σ by INSTANT(σ).

► **Theorem 10.** INSTANT has $O(1)$ -RoE for light instances.

The theorem follows immediately from the following two lemmas (due to space limit, see the full version [57] for their proofs).

► **Lemma 11.** If (T, λ) is light, then $\mathbb{E}[\text{cost}(\text{INSTANT}(\sigma)) \mid \sigma \sim (T, \lambda)^\tau] = \tau \cdot \pi(T, \lambda)$.

► **Lemma 12.** If (T, λ) is light, then $\mathbb{E}[\text{cost}(\text{OPT}(\sigma)) \mid \sigma \sim (T, \lambda)^\tau] = \Omega(1) \cdot \tau \cdot \pi(T, \lambda)$.

We now turn our attention to heavy instances. An instance (T, λ) is *heavy* if for every subtree $T' \subseteq T$, we have $\pi(T', \lambda) > 1$. By monotonicity of $\pi(\cdot, \lambda)$, we obtain the following equivalent definition. Recall that for a vertex $u \in V(U)$, w_u denotes the weight of the edge incident to u on the path from $\gamma(T)$ to u .

► **Definition 13.** A stochastic MLA instance (T, λ) is called *heavy* if $w_u \geq 1/\lambda(u)$ for all $u \in V(T)$ with $\lambda(u) > 0$.

To give some intuition, suppose that u is a vertex of a heavy instance, and r and r' are two consecutive (random) requests located on u . Then, the expected duration between their arrival times is $1/\lambda(u) < w_u$. This suggests that to minimize the cost, we should, on average, gather r and r' into the same service in order to avoid paying twice the weight cost w_u . Since we expect services to serve a group of two or more requests, our stochastic assumptions suggest that the services must follow some form of regularity.

In Section 5, we present an algorithm called PLAN, that given a heavy instance (T, λ) , computes for each vertex $u \in V(T)$ a period $p_u > 0$, and will serve u at every time that is a multiple of p_u . One intuitive property of these periods $\{p_u : u \in V(T)\}$ is that the longer the distance to the root, the longer the period. While losing only a constant fraction of the expected cost, we choose the periods to be (scaled) powers of 2. This enables us to optimize the weights of the services in the long run. One interesting feature of our algorithm is that it acts “blindly”: the algorithm does not need to know the requests, but only the arrival rate of each point. Indeed, our algorithm may serve a vertex where there are no pending requests. For the details of the PLAN algorithm, see Section 5.

► **Theorem 14.** *PLAN has $O(1)$ -RoE for heavy instances.*

We remark that light instances and heavy instances are not complementary: there are instances that are neither light nor heavy.³ In Section 6, we focus on the general case of arbitrary instances. The strategy here is to partition the tree (and the sequence of requests) into two groups of vertices (two groups of requests) so that the first group corresponds to a light instance where we can apply the instant strategy while the second group corresponds to a heavy instance where we can apply a periodic strategy. However, this correspondence for the heavy group is not straightforward. For this, we need to define an *augmented tree* that is a copy of the original tree, with the addition of some carefully chosen vertices. Each new vertex is associated with a subset of vertices of the original tree called *part*. We then define an arrival rate for each of these new vertices that is equal to the sum of the arrival rates of the vertices in the corresponding part. We show that this defines a heavy instance on which we can apply the algorithm PLAN. For each service made by PLAN on each of these new vertices, we serve all the pending requests in the corresponding part. The full description of this algorithm, called GEN, is given in Section 6. We show that this algorithm achieves a constant ratio of expectations.

► **Theorem 15.** *GEN has $O(1)$ -RoE for arbitrary stochastic instances.*

5 Heavy instances

In this section, we analyze heavy MLA instances. Recall that an instance (T, λ) is called heavy if $w_u \geq 1/\lambda(u)$ for all $u \in V(T)$ with $\lambda(u) > 0$. To serve this type of MLA instances, we devise an algorithm PLAN and prove that it achieves a constant ratio of expectations. Our approach can be seen as a generalization of the periodical strategy for a single-edge case. Once again, we serve the requests periodically, although this time, we may assign different periods for different vertices. Intuitively, vertices closer to the root and having a greater arrival rate should be served more frequently. For this reason, PLAN generates a partition P of a given tree T into a family of subtrees (clusters) and assigns them specific periods.

The partition procedure allows us to analyze each cluster separately. Thus, it is sufficient to estimate the performance of PLAN algorithm when restricted to a given subtree $T' \in P$. To lower bound the cost generated by OPT on T' , we split the weight of T' among its vertices using a saturation procedure. Then we say that for each vertex v , the optimal algorithm either covers the delay cost of all the requests arriving at v within a given time horizon or it pays some share of the service cost. The last step is to round the periods assigned to the subtrees in P to minimize the cost of PLAN. In what follows, we present the details.

Periodical algorithm PLAN

As mentioned before, the main idea is to split tree T rooted at vertex γ into a family of subtrees and serve each of them periodically. In other words, we aim to find a partition $P = \{T_1, T_2, \dots, T_k\}$ of T where each subtree T_i besides the one containing γ is rooted at the leaf vertex of another subtree. At the same time, we assign each subtree T_i some period p_i . To decide how to choose the values of p_i s, recall how we picked the period for a single-edge case. In that setting, for the period p , we had an equality between the expected delay cost $\lambda/2 \cdot p^2$ at the leaf u and the weight w of the edge. Thus, the intuition behind the PLAN algorithm is as follows.

³ There exists a stochastic MLA instance where the ratio of expectations are both unbounded if INSTANT or PLAN is directed applied to deal with. See the appendix in the full version [57] for details.

We start by assigning each vertex $v \in T$ a process that saturates the edge connecting it to the parent at the pace of $\lambda(v)/2 \cdot t^2$, i.e., within the time interval $[t, t + \epsilon]$ it saturates the weight of $\lambda(v)/2 \cdot ((t + \epsilon)^2 - t^2)$. By saturation, here we mean assigning a part of the edge weight to the vertex. In other words, at time t each vertex v has a budget that is equal to its expected delay cost, i.e., $\lambda(v)/2 \cdot t^2$, and uses it to cover some part of the edge weight. Whenever an edge gets saturated, the processes that contributed to this outcome start working together with the processes that are still saturating the closest ancestor edge. As the saturation procedure within the whole tree T reaches the root γ , we cluster all the vertices corresponding to the processes that made it possible into the first subtree T_1 . Moreover, we set the period of T_1 to the time it got saturated. After this action, we are left with a partially saturated forest having the leaves of T_1 as the root vertices. The procedure, however, follows the same rules, splitting the forest further into subtrees T_2, \dots, T_k .

To simplify the formal description of our algorithm, we first introduce some new notations. Let $p(v)$ denote the saturation process defined for a given vertex v . As mentioned before, we define it to saturate the parent edge at the pace of $\lambda(v)/2 \cdot t^2$. Moreover, we extend this notation to the subsets of vertices, i.e., we say that $p(S)$ is the saturation process where all the vertices in S cooperate to cover the cost of an edge. The pace this time is equal to $\lambda(S)/2 \cdot t^2$. To trace which vertices cooperate at a given moment and which edge they saturate, we denote the subset of vertices that v works with by $S(v)$ and the edge they saturate by $e(v)$. We also define a method $\text{join}(u, v)$ that takes as the arguments two vertices and joins the subsets they belong to. It can be called only when the saturation process of $S(u)$ reaches v . Formally, at this moment, the join method merges subset $S(u)$ with $S(v)$ and sets $e(v)$ as the outcome of the function e on all the vertices in the new set. It also updates the saturation pace of the new set. We present the pseudo-code for PLAN as Algorithm 1 and an example as a visual support shown in Figure 1, followed by some properties of the partition generated by this algorithm (see the appendix in the full version [57] for the detailed proof) and the lower bounding scheme (Lemma 17).

► **Proposition 16.** *Let (T, λ) be a heavy instance and let $P = \{T_1, T_2, \dots, T_k\}$ be the partition generated on it by Algorithm 1. We denote the period corresponding to T_i by p_i . Assuming that T_i s are listed in the order they were added to P , it holds that:*

1. each T_i is a rooted subtree of T ;
2. the periods are increasing, i.e., $1 \leq p_1 \leq p_2 \leq \dots \leq p_k$;
3. each vertex $v \in T_i$ saturated exactly $\lambda(v)/2 \cdot p_i^2$ along the path to the root of T_i .

► **Lemma 17.** *Let (T, λ) be a heavy instance. We denote the partition generated for it by Algorithm 1 by $P = \{T_1, T_2, \dots, T_k\}$ and the period corresponding to T_i by p_i for all $i \in [k]$. Let T_i be any subtree in P , and let us define σ_i as a random sequence of requests arriving within the MLA instance restricted to T_i over a time horizon τ . We assume that τ is a multiple of p_i . It holds that $\mathbb{E}[\text{cost}(\text{OPT}(\sigma_i), T_i) \mid \sigma_i \sim (T_i, \lambda|_{T_i})^\tau] \geq \frac{3}{16} \cdot w(T_i) \cdot \frac{\tau}{p_i}$.*

The main idea is to use the same approach as in Section 3 and lower bound the cost incurred by OPT within a shorter time interval. The proof of Lemma 17 can be found in the full version [57].

PLAN has $\text{RoE} \leq 64/3 = 21.34$ for heavy instances

In Algorithm 1, each subtree T_i is served periodically with periods p_i . In this setting, to serve any cluster besides the one containing the root vertex γ , not only do we need to cover the service cost of the cluster vertices but also the cost of the path connecting them to γ . Since

Algorithm 1 PLAN (part I).

Input: an heavy instance (T, λ) with tree T rooted at γ
Output: a partition $P = \{T_1, T_2, \dots, T_k\}$ of T , each subtree T_i assigned a period p_i

- 1 let R be the set of roots, initially $R = \{\gamma\}$
- 2 **for** each vertex $v \in V(T)$ **do**
- 3 define the saturation process $p(v)$ as described before
- 4 set $S(v) := \{v\}$ and $e(v) := \text{par}(v)$
- 5 **end**
- 6 start the clock at time 0
- 7 **while** there exist some unclustered vertices in T **do**
- 8 wait until the first time t_e when an edge $e = (u, v)$ gets saturated
- 9 **if** $v \notin R$ **then**
- 10 join(u, v)
- 11 **end**
- 12 **else**
- 13 add cluster $C := S(u) \cup \{v\}$ to partition P
- 14 set the period p for C to be equal to t_e
- 15 set the saturation pace for C to 0
- 16 extend R by the leaves in C
- 17 **end**
- 18 **end**

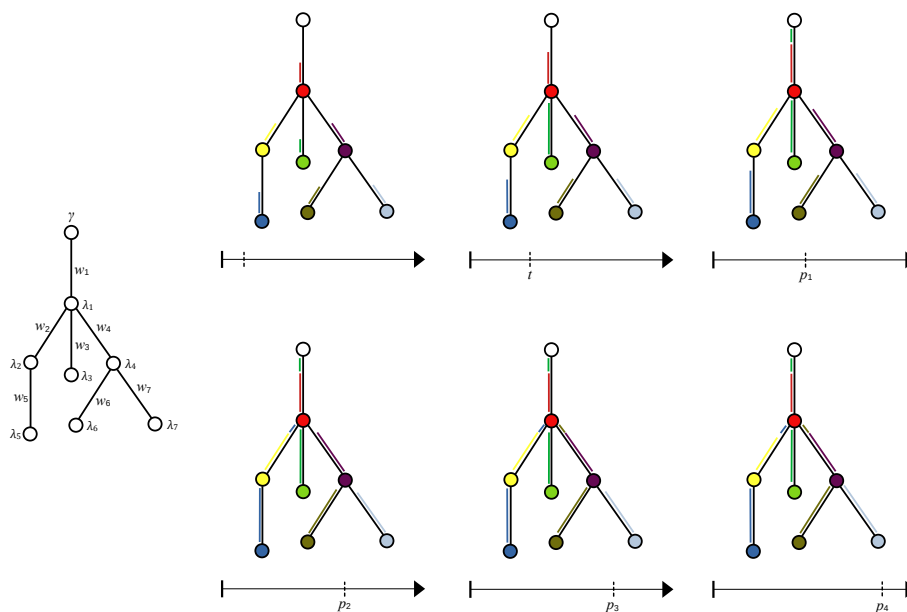


Figure 1 Here is an example to show how Algorithm 1 works on an heavy instance. Given the tree consisting of 7 vertices (with $w_i \geq 1/\lambda_i$ for each vertex $i \in [7]$ marked in different color), we use the length of the colored line to denote the saturated amount (i.e., $\lambda_i/2 \cdot t^2$) of a vertex i at any time t . At time p_1 , the subtree T_1 including vertices 1 and 3 is determined; similarly, T_2 includes vertices 2 and 5 at time t_2 ; T_3 includes vertices 4 and 6 at time p_3 ; and T_4 includes vertex 7 at time p_4 .

we only know how to lower bound the cost incurred by OPT on the clusters, we improve the PLAN algorithm to get rid of this issue. The idea is to round the periods p_i s to be of form $2^{e_i}p_1$ for some positive integers e_i . Thus, whenever we need to serve some cluster S_i , we know that we get to serve all the clusters generated before it as well. Due to the space limits, the formal pseudo-codes of the rounding procedures (Algorithm PLAN, part 2) and serving a random sequence of requests (Algorithm PLAN, part 3) can be found in the full version [57].

Let (T, λ) be a heavy instance and let $P = \{T_1, \dots, T_k\}$ be the partition generated for it by Algorithm 1. Let (p_1, \dots, p_k) and $(\hat{p}_1, \dots, \hat{p}_k)$ denote the periods obtained from Algorithm 1 and rounded periods (by Algorithm PLAN, part 2), respectively. Now we analyze the cost of PLAN on $\sigma \sim (T, \lambda)^\tau$, where the time horizon τ is a multiple of $2\hat{p}_k$. Since we align the periods to be of form $2^l \hat{p}_1$ for some positive integer l , whenever PLAN serves some tree T_i , it serves all the trees containing the path from T_i to γ at the same time. Thus, the service cost can be estimated on the subtree level. Moreover, since for each $i \in [k]$ we round p_i such that $\hat{p}_i \leq p_i$, the expected delay cost incurred within $[0, \hat{p}_i]$ does not exceed $w(T_i)$. Denoting by $\sigma_i \sim (T_i, \lambda|_{T_i})^{\hat{p}_i}$ for $i \in [k]$, we have

$$\mathbb{E}[\text{cost}(\text{PLAN}(\sigma), T)] = \sum_{i=1}^k \frac{\tau}{\hat{p}_i} \cdot \mathbb{E}[\text{cost}(\text{PLAN}(\sigma_i), T_i)] = \sum_{i=1}^k \frac{\tau}{\hat{p}_i} \cdot 2w(T_i).$$

On the other hand, the expected cost of OPT for σ , i.e., $\mathbb{E}[\text{cost}(\text{OPT}(\sigma), T)]$, is at least

$$\sum_{i=1}^k \mathbb{E}[\text{cost}(\text{OPT}(\sigma|_{T_i}), T_i)] \geq \sum_{i=1}^k \frac{\tau}{p_i} \frac{3}{16} w(T_i).$$

By definition, it holds that $p_i < 2\hat{p}_i$ for $i \in [k]$. We can rewrite the above as

$$\mathbb{E}[\text{cost}(\text{OPT}(\sigma), T)] > \sum_{i=1}^k \frac{\tau}{2\hat{p}_i} \frac{3}{16} w(T_i) = \frac{3}{32} \sum_{i=1}^k \frac{\tau}{\hat{p}_i} w(T_i),$$

and hence establishing $\text{RoE}(\text{PLAN}) = 64/3$.

6 General instances

Now we devise an algorithm GEN for an arbitrary stochastic instance (T, λ) , which achieves a constant ratio of expectations. The main idea is to distinguish two types of requests and apply a different strategy for each type. The first type is the requests that are located close to the root. These requests will be served immediately at their arrival times, i.e., we apply INSTANT to the corresponding sub-sequence. The second type includes all remaining requests, and they are served in a periodic manner. To determine the period of these vertices, we will use the algorithm PLAN on a specific heavy instance (T', λ^h) . The construction of this heavy instance relies on a partition of the vertices of T into *balanced parts*. Intuitively, a part is balanced when it is light (or close to being light), but if we merge all vertices of the part into a single vertex whose weight corresponds to the average distance to the root of the part, then we obtain a heavy edge. This “merging” process is captured by the construction of the *augmented tree* T' , which is part of the heavy instance. The augmented tree is essentially a copy of T with the addition of one (or two) new vertices for each balanced part.

Once determining the corresponding heavy instance, we can compute the periods of each vertex of the heavy instance using PLAN. The vertex period in the original instance is equal to the corresponding vertex period in the heavy instance. For the full description of GEN, see Algorithm 2 in the appendix of the full version [57]. The main challenge is to analyze the ratio of expectations and in particular, to establish good lower bounds on the expected cost of the optimal offline schedule. Due to the space limits, check the full version [57], where we prove two lower bounds that depend on the heaviness of each part of the balanced partition.

Notations and additional assumptions. Given the edge-weighted tree T rooted at γ and a set of vertices $U \subseteq V(T)$, $T[U]$ denotes the forest induced on U in T . We say that a subset $U \subseteq V(T)$ is *connected* if $T[U]$ is connected (i.e., $T[U]$ is a subtree of T but not a forest). If $U \subseteq V(T)$ is connected, we write $\gamma(U) = \gamma(T[U])$ to denote the root vertex of $T[U]$, i.e., the vertex in U which has the shortest path length to γ in the original tree T . Given any vertex $u \in V(T)$, let $V_u \subseteq V(T)$ denote all the descendant vertices of u in T (including u). For simplicity, set $w_{\gamma(T)} = \infty$. Given $T = (V, E)$, $\lambda : V(T) \rightarrow \mathbb{R}_+$ and $U \subseteq V$, we denote $\lambda|_U : U \rightarrow \mathbb{R}_+$ such that $\lambda|_U(u) = \lambda(u)$ for each $u \in U$. For a sequence of requests $\sigma \sim (T, \lambda)$, we use $\sigma|_U = \{r \in \sigma \mid \ell(r) \in U\}$ to denote the corresponding sequence for $T[U]$. In this section, we assume $\lambda(\gamma) = 0$ and γ has only one child.

Balanced partition of $V(T)$. Recall that $\pi(T, \lambda) = \sum_{u \in V(T)} \lambda(u) \cdot d(u, \gamma(T))$. When the context is clear we simply write $\pi(T) = \pi(T, \lambda)$, and for a connected subset $U \subseteq V(T)$ we simply write $\pi(U) := \pi(T[U], \lambda|_U)$.

► **Definition 18.** Given a stochastic instance (T, λ) , we say that $U \subseteq V(T)$ is balanced if U is connected and if one of the following conditions holds:

- (1) U is of type-I: $\pi(U) \leq 1$, and either $\gamma(U) = \gamma(T)$ or $\pi(U \cup \{\text{par}(\gamma(U))\}) > 1$;
- (2) U is of type-II: $\pi(U) > 1$, and for each child vertex y of $\gamma(U)$ in $T[U]$, we have $\pi(\{\gamma(U)\} \cup (U \cap V_y)) < 1$.

Remark that the root $\gamma(U)$ of a balanced type-II part U must have at least two children in $T[U]$.

► **Definition 19.** Given a stochastic instance (T, λ) and a partition \mathcal{P} of the vertices $V(T)$, we say that \mathcal{P} is a balanced partition of tree T if every part $U \in \mathcal{P}$ is balanced.

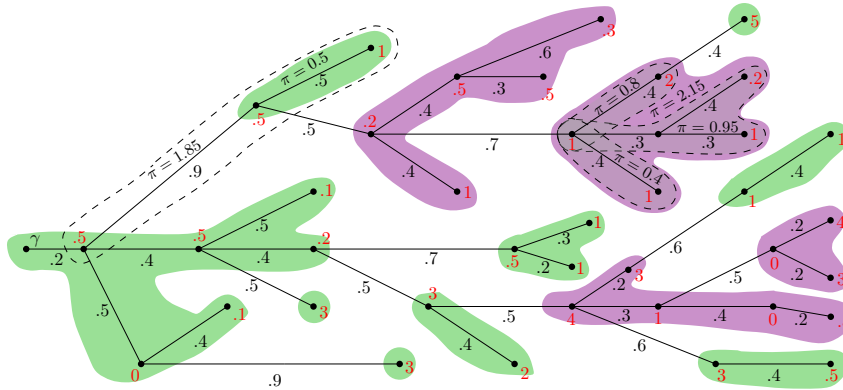
See Figure 2 for an example of a balanced partition. If \mathcal{P} is a *balanced partition* of T , then the part $U \in \mathcal{P}$ containing $\gamma(T)$ is called the *root part* in \mathcal{P} . Since we assume that $\gamma(T)$ has only one child vertex, we deduce from the previous remark that the root part is necessarily of type-I. Given a balanced partition \mathcal{P} , we denote $\mathcal{P}^* := \mathcal{P} \setminus \{\gamma(\mathcal{P})\}$; $\mathcal{P}_1 \subseteq \mathcal{P}$ the set of type-I parts; $\mathcal{P}_1^* := \mathcal{P}_1 \cap \mathcal{P}^*$ and $\mathcal{P}_2 \subseteq \mathcal{P}$ the set of type-II parts.

► **Lemma 20.** Given any stochastic instance (T, λ) , there exists a balanced partition of T . Moreover, such a partition can be computed in $O(|V(T)|^2)$ time.

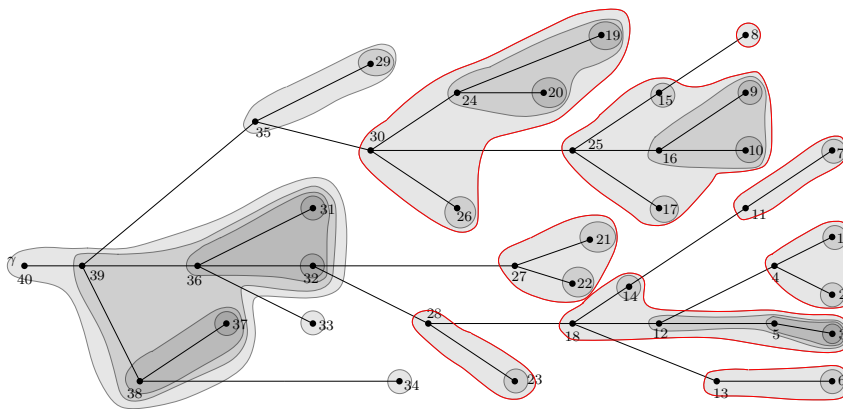
The algorithm to construct a partition \mathcal{P} works as follows. We order the vertices u_1, \dots, u_n by decreasing distances from the root, i.e., for $1 \leq i < j$, $d(u_i, \gamma(T)) \geq d(u_j, \gamma(T))$. Let $\mathcal{P}^{(0)} = \emptyset$. For each $i \in [n]$, let $C_i \subseteq \{1, \dots, n-1\}$ be the subset of indexes j s.t. (i) u_j is a child of u_i ; (ii) $u_j \notin \bigcup_{U \in \mathcal{P}^{(i-1)}} U$. Define $U_i := (\bigcup_{j \in C_i} U_j) \cup \{u_i\}$ recursively. If $i = n$ (i.e., if u_i is the root of T) or $\pi(U_i \cup \{\text{par}(u_i)\}) > 1$, then define $\mathcal{P}^{(i)} := \mathcal{P}^{(i-1)} \cup \{U_i\}$. Otherwise, define $\mathcal{P}^{(i)} := \mathcal{P}^{(i-1)}$. See Figure 3 for a visual support.

The heavy instance. Given a stochastic instance (T, λ) , and a balanced partition \mathcal{P} of T , we construct a tree T' that we call the *augmented tree* of T . This tree is essentially a copy of T with additional one or two vertices for each part of \mathcal{P}^* .⁴ Then, we define arrival rates λ^h on T' in a way that the stochastic instance (T', λ^h) is heavy. Finally, we construct from a request sequence σ , the corresponding *heavy sequence* σ^h for the augmented tree.

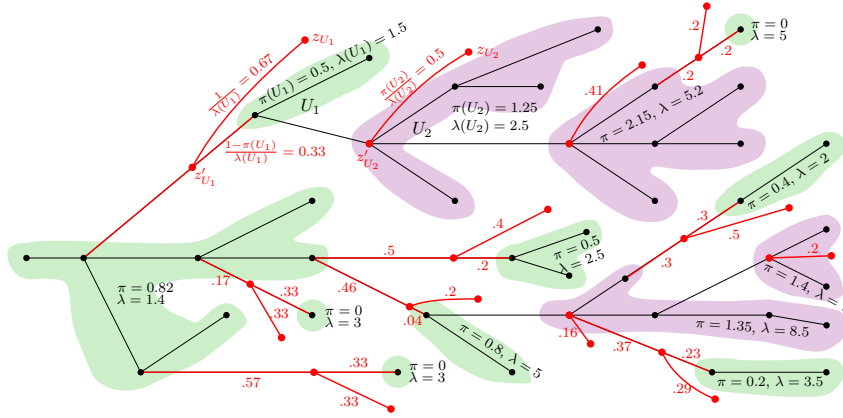
⁴ Recall that $\mathcal{P}^* = \mathcal{P} \setminus \{\gamma(\mathcal{P})\}$, where $\gamma(\mathcal{P})$ denotes the particular part in \mathcal{P} including the tree root $\gamma(T)$.



■ **Figure 2** An example of a balanced partition (Definition 19). The weight of each edge is shown in black, and the arrival rate of each vertex is shown in red. Green subsets corresponds to parts of type-I while purple ones correspond to parts of type-II. Some value of π are shown for the top-left type-I part and for the top-right type-II part.



■ **Figure 3** Construction of a balanced partition in the proof of Lemma 20. The weights of the edges and the arrival rates of the vertices are the same as in Figure 2. The numbers represent an ordering of the vertices. The gray sets corresponds to U_i , for $i \in [40]$. We illustrate the step $i = 30$ of the algorithm. We have $C_{30} = \{24, 26\}$ and $U_{30} = \{u_{30}\} \cup U_{24} \cup U_{30} = \{u_{30}, u_{24}, u_{19}, u_{20}, u_{26}\}$. Since $\pi(U_{30} \cup \{u_{25}\}) = 2.65 > 1$, we add U_{30} into $\mathcal{P}^{(29)}$ to create $\mathcal{P}^{(30)}$ (red stroke). We remark that U_{30} is a balanced subset of type-II and U_{27} is a balanced subset of type-II, while U_{25} is not a balanced subset.



■ **Figure 4** The construction of the augmented tree associated with the instance and the balanced partition of Figure 2. The new edges and vertices are shown in red. The illustrate the calculation of the length of these edges for a part U_1 of type-I and for a part U_2 of type-II. For each part U of the partition, we indicate the values of $\lambda(U)$ and $\pi(U)$. For simplicity, we have rounded the values to their second decimal.

To construct the augmented tree, define $T' = (V', E')$ where $V' = V(T) \cup \{z_U, z'_U : U \in \mathcal{P}^*\}$, and the edge set E' is constructed based on $E(T)$ as follows. First, for each $U \in \mathcal{P}_1^*$, replace the edge $(\gamma(U), \text{par}(\gamma(U)))$ of length $w_{\gamma(U)}$ by two edges $(\gamma(U), z'_U)$ and $(z'_U, \text{par}(\gamma(U)))$ of respective lengths $(1 - \pi(U))/\lambda(U)$ and $w_{\gamma(U)} - (1 - \pi(U))/\lambda(U)$, where $\text{par}(\gamma(U))$ denotes the parent of $\gamma(U)$ in T . Then, add an edge (z_U, z'_U) of weight $1/\lambda(U)$. Finally, for each $U \in \mathcal{P}_2$, set $z'_U = \gamma(U)$, and add an edge (z_U, z'_U) of weight $\pi(U)/\lambda(U)$.

This completes the construction of the augmented tree (see Figure 4 for visual support). Note that if a part $U = \{u\}$ in \mathcal{P} contains only one vertex, then we have $\pi(U) = 0$, and thus part U is necessarily of type-I. To simplify, in the following, we identify vertices in T with their copy in T' and consider that $V(T)$ is a subset of $V(T')$. For the arrival rates of the heavy instance, recall that $\mathcal{P}^* = \mathcal{P} \setminus \{\gamma(\mathcal{P})\}$, where $\gamma(\mathcal{P})$ denotes the part in \mathcal{P} containing the root $\gamma(T)$. We define $\lambda^h : V(T') \rightarrow \mathbb{R}_+$ as follows: for each $U \in \mathcal{P}^*$, set $\lambda^h(z_U) = \lambda(U)$; and $\lambda^h(u) = 0$ otherwise.⁵

► **Definition 21.** Given a stochastic instance (T, λ) , a balanced partition \mathcal{P} of tree T , the corresponding augmented tree T' , and a sequence of request $\sigma \sim (T, \lambda)^\tau$, we construct the heavy sequence associated with σ for T' and denoted by σ^h as follows: for each request $r = (u, t) \in \sigma$ located on some part $U \in \mathcal{P}^*$ (i.e., $u \in U$), there is a request (z_U, t) in σ^h .

The algorithm GEN. The input is a stochastic instance (T, λ) , known in advance, and a sequence of requests σ for T , revealed over time. In the pre-processing step, GEN computes a balanced partition \mathcal{P} of T (Lemma 20), a light instance $(T[\gamma(\mathcal{P})], \sigma|_{\gamma(\mathcal{P})})$, and the heavy instance (T', σ^h) . At each request arrival, GEN updates the sequences of requests $\sigma|_{\gamma(\mathcal{P})}$ and σ^h . The algorithm runs PLAN (Algorithm 1) on input (T', σ^h) . Suppose that PLAN serves at time t a set of vertices $\{z_U, U \in \mathcal{P}'\} \subseteq V(T')$ for some subset $\mathcal{P}' \subseteq \mathcal{P}^*$. Then, GEN serves at time t all pending requests on vertices $(\bigcup_{U \in \mathcal{P}'} U) \subseteq V(T)$.

⁵ It is important to notice that σ^h can be constructed in an online fashion: for any time t , the restriction of the σ^h to the requests that arrives before t only depends on the requests that arrives before t in σ .

In parallel, the algorithm runs INSTANT on input $(T[\gamma(\mathcal{P})], \sigma|_{\gamma(\mathcal{P})})$, and performs the same services. This finishes the description of the algorithm GEN see the formal pseudo-code of GEN (Algorithm 2) and a visual support in Figures 2, 3 and 4. The detailed proof of GEN achieving a constant ratio of expectations can be found in the appendix of the full version [57].

■ **Algorithm 2** GEN.

Input: stochastic instance (T, λ) and $\sigma \sim (T, \lambda)^\tau$
Output: a valid schedule of σ

- 1 — pre-processing the given instance —
- 2 produce a balanced partition \mathcal{P} for T (see Lemma 20);
- 3 construct the heavy instance (T', λ^h) ;
- 4 use PLAN (Algorithm 1) to determine the period of the vertices of T' ;
- 5 — Serve the requests —
- 6 **for** each request $r \in \sigma$ **do**
- 7 **if** r arrives in $\gamma(\mathcal{P})$ **then**
- 8 serve r immediately.
- 9 **end**
- 10 **if** r arrives in a vertex of $U \in \mathcal{P}^*$ **then**
- 11 serve r at time $t(r')$ where $r' \in \sigma^h$ is the corresponding request located on z_U
 and $t(r')$ is the time at which r' is served by PLAN(σ^h).
- 12 **end**
- 13 **end**

7 Other related works

The MLA problem was first introduced by Bienkowski et al. [16] and they study a more general version in their paper, where the cost of delaying a request r by a duration t is $f_r(t)$. Bienkowski et al. proposed an $O(d^4 2^d)$ -competitive online algorithm for this general delay cost version problem, where d denotes the depth of the given tree. A deadline version of MLA is also considered in [16], where each request r has a time window (between its arrival and its deadline) and it has to be served no later than its deadline. The target is to minimize the total service cost for serving all the requests. For this deadline version problem, they proposed an online algorithm with a better competitive ratio of $d^2 2^d$. Later, the competitiveness of MLA was further improved to $O(d^2)$ [13] for the general delay cost version and to $O(d)$ [27, 58] for the deadline version. However, for the delay cost version, no matching lower bound has been found thus far – the current best lower bound on MLA with delays is only 4 [16, 17, 18], restricted to a path case with linear delays. In the offline setting, MLA is NP-hard in both delay and deadline versions [3, 15], and a 2-approximation algorithm was proposed by Becchetti et al. [15] for the deadline version. For a special path case of MLA with the linear delay, Bienkowski et al. [22] proved that the competitiveness is between 3.618 and 5, improving on an earlier 8-competitive algorithm given by Brito et al. [26]. Thus far, no previous work has studied MLA in the stochastic input model, no matter the delay or deadline versions.

Two special cases of MLA with linear delays, one called TCP-acknowledgment ($d = 1$) and one called Joint Replenishment (abbr. JRP, $d = 2$) are of particular interests: TCP-acknowledgment (a.k.a. single item lot-sizing problem, [25, 41, 61, 29, 44]) models the data

transmission issue from sensor networks [68, 51], while JRP models the inventory control issue from supply chain management [5, 37, 42, 64, 48]. For TCP-acknowledgment, in the online setting there exists an optimal 2-competitive deterministic algorithm [32] and an optimal $e/(e-1)$ -competitive randomized algorithm [45, 63]; in the offline setting, the problem can be solved in $O(n \log n)$ time, n denoting the number of requests [1]. For JRP, the competitiveness is between 3 [28] and 2.754 [21]; in the offline setting, JRP is NP-hard [3] and also APX-hard [60, 20]. The current best approximation ratio for JRP is 1.791 [53, 54, 52, 21]. For a deadline version of JRP, Bienkowski et al. [21] proposed an optimal 2-competitive algorithm.

Another problem, called online service with delays (OSD), first introduced by Azar et al. [9], is closely related to MLA (with linear delays). In this OSD problem, a n -points metric space is given as input. The requests arrive at metric points over time, and a server is available to serve the requests. The target is to serve all the requests in an online manner such that their total delay cost plus the total distance traveled by the server is minimized. Note that MLA can be seen as a special case of OSD when the given metric is a tree, and the server has to always come back to a particular tree vertex immediately after serving some requests elsewhere. For OSD, Azar et al. [9] proposed an $O(\log^4 n)$ -competitive online algorithm in their paper. Later, the competitive ratio for OSD is improved from $O(\log^2 n)$ (by Azar and Touitou [13]) to $O(\log n)$ (by Touitou [67]).

Recently, many other online problems with delays/deadline have also drawn a lot of attention besides MLA, such as online matching with delays [33, 6, 4, 24, 23, 34, 10, 31, 55, 12, 59, 56, 49], online service with delays [9, 13, 66, 67], facility location with delays/deadline [19, 13, 14], Steiner tree with delays/deadline [14], bin packing with delays [8, 35, 36, 2], set cover with delays [7, 65, 50], paging with delays/deadline [38, 39], list update with delays/deadline [11], and many others [59, 30, 66, 40, 43, 46].

8 Concluding remarks

In this paper, we studied MLA with additional stochastic assumptions on the sequence of the input requests. In the following, we briefly discuss some potential future directions.

Does the greedy algorithm achieve a constant ratio of expectations? An intuitive heuristic algorithm for MLA is *Greedy*, which works as follows: *each time when a set of requests R arriving at vertices $U \subseteq V(T)$ have the total delay cost equal to the weight of the minimal subtree of T including γ and U , serve all the requests R .* Does this greedy algorithm achieve a constant ratio of expectations?

Is it possible to generalize MLA with edge capacity and k tree roots? One practical scenario on MLA is that each edge has a capacity on the maximum number of requests served in one service if this edge is used, such as [61, 44, 64]. We conjecture that some $O(1)$ -RoE online algorithm can be proposed for this generalized MLA with edge capacity. Another generalized version of MLA is to assume k tree roots available for serving requests concurrently. That is, a set of pending requests can be served together by connecting to any of k servers. The question is, how to design an online algorithm for this k -MLA problem? Does there exist $O(1)$ -RoE algorithm still?

What about the other online network design problems with delays in the Poisson arrival model? Recall that the online problems of service with delays (and its generalization called k -services with delays), facility location with delays, Steiner tree/forest with delays are all closely related to MLA. Does there exist online algorithm with $O(1)$ -RoE for each problem?

References

- 1 Alok Aggarwal and James K. Park. Improved algorithms for economic lot size problems. *Operations research*, 41(3):549–571, 1993. doi:10.1287/OPRE.41.3.549.
- 2 Lauri Ahlroth, André Schumacher, and Pekka Orponen. Online bin packing with delay and holding costs. *Operations Research Letters*, 41(1):1–6, 2013. doi:10.1016/J.ORL.2012.10.006.
- 3 Esther Arkin, Dev Joneja, and Robin Roundy. Computational complexity of uncapacitated multi-echelon production planning problems. *Operations research letters*, 8(2):61–66, 1989.
- 4 Itai Ashlagi, Yossi Azar, Moses Charikar, Ashish Chiplunkar, Ofir Geri, Haim Kaplan, Rahul Makhijani, Yuyi Wang, and Roger Wattenhofer. Min-cost bipartite perfect matching with delays. In *Proc. APPROX / RANDOM*, pages 1:1–1:20, 2017. doi:10.4230/LIPICS.APPROX-RANDOM.2017.1.
- 5 Y. Askoy and S. S. Erenguk. Multi-item inventory models with coordinated replenishment: a survey. *International Journal of Operations and Production Management*, 8:63–73, 1988.
- 6 Yossi Azar, Ashish Chiplunkar, and Haim Kaplan. Polylogarithmic bounds on the competitiveness of min-cost perfect matching with delays. In *Proc. SODA*, pages 1051–1061, 2017. doi:10.1137/1.9781611974782.67.
- 7 Yossi Azar, Ashish Chiplunkar, Shay Kutten, and Noam Touitou. Set cover with delay—clairvoyance is not required. In *Proc. ESA*, pages 8:1–8:21, 2020. doi:10.4230/LIPICS.ESA.2020.8.
- 8 Yossi Azar, Yuval Emek, Rob van Stee, and Danny Vainstein. The price of clustering in bin-packing with applications to bin-packing with delays. In *Proc. SPAA*, pages 1–10, 2019. doi:10.1145/3323165.3323180.
- 9 Yossi Azar, Arun Ganesh, Rong Ge, and Debmalya Panigrahi. Online service with delay. In *Proc. STOC*, pages 551–563, 2017. doi:10.1145/3055399.3055475.
- 10 Yossi Azar and Amit Jacob-Fanani. Deterministic min-cost matching with delays. *Theory of Computing Systems*, 64(4):572–592, 2020. doi:10.1007/S00224-019-09963-7.
- 11 Yossi Azar, Shahar Lewkowicz, and Danny Vainstein. List update with delays or time windows. In *Proc. ICALP*, pages 15:1–15:20, 2024. doi:10.4230/LIPICS.ICALP.2024.15.
- 12 Yossi Azar, Runtian Ren, and Danny Vainstein. The min-cost matching with concave delays problem. In *Proc. SODA*, pages 301–320, 2021. doi:10.1137/1.9781611976465.20.
- 13 Yossi Azar and Noam Touitou. General framework for metric optimization problems with delay or with deadlines. In *Proc. FOCS*, pages 60–71, 2019. doi:10.1109/FOCS.2019.00013.
- 14 Yossi Azar and Noam Touitou. Beyond tree embeddings—a deterministic framework for network design with deadlines or delay. In *Proc. FOCS*, pages 1368–1379, 2020. doi:10.1109/FOCS46700.2020.00129.
- 15 Luca Becchetti, Alberto Marchetti-Spaccamela, Andrea Vitaletti, Peter Korteweg, Martin Skutella, and Leen Stougie. Latency-constrained aggregation in sensor networks. *ACM Transactions on Algorithms*, 6(1):1–20, 2009. doi:10.1145/1644015.1644028.
- 16 Marcin Bienkowski, Martin Böhm, Jaroslaw Byrka, Marek Chrobak, Christoph Dürr, Lukáš Folwarczný, Lukasz Jež, Jiří Sgall, Nguyen Kim Thang, and Pavel Veselý. Online algorithms for multi-level aggregation. In *Proc. ESA*, pages 12:1–12:17, 2016.
- 17 Marcin Bienkowski, Martin Böhm, Jaroslaw Byrka, Marek Chrobak, Christoph Dürr, Lukáš Folwarczný, Lukasz Jež, Jiří Sgall, Nguyen Kim Thang, and Pavel Veselý. Online algorithms for multilevel aggregation. *Operations Research*, 68(1):214–232, 2020. doi:10.1287/OPRE.2019.1847.
- 18 Marcin Bienkowski, Martin Böhm, Jaroslaw Byrka, Marek Chrobak, Christoph Dürr, Lukáš Folwarczný, Lukasz Jež, Jiří Sgall, Nguyen Kim Thang, and Pavel Veselý. New results on multi-level aggregation. *Theoretical Computer Science*, 861:133–143, 2021. doi:10.1016/J.TCS.2021.02.016.
- 19 Marcin Bienkowski, Martin Böhm, Jaroslaw Byrka, and Jan Marcinkowski. Online facility location with linear delay. In *Proc. APPROX/RANDOM*, pages 45:1–45:17, 2022. doi:10.4230/LIPICS.APPROX/RANDOM.2022.45.

- 20 Marcin Bienkowski, Jarosław Byrka, Marek Chrobak, Neil Dobbs, Tomasz Nowicki, Maxim Sviridenko, Grzegorz Świrszcz, and Neal E. Young. Approximation algorithms for the joint replenishment problem with deadlines. *Journal of Scheduling*, 18(6):545–560, 2015. doi:10.1007/S10951-014-0392-Y.
- 21 Marcin Bienkowski, Jarosław Byrka, Marek Chrobak, Łukasz Jeż, Dorian Nogneng, and Jiří Sgall. Better approximation bounds for the joint replenishment problem. In *Proc. SODA*, pages 42–54, 2014.
- 22 Marcin Bienkowski, Jarosław Byrka, Marek Chrobak, Łukasz Jeż, Jiří Sgall, and Grzegorz Stachowiak. Online control message aggregation in chain networks. In *Proc. WADS*, pages 133–145, 2013.
- 23 Marcin Bienkowski, Artur Kraska, Hsiang-Hsuan Liu, and Paweł Schmidt. A primal-dual online deterministic algorithm for matching with delays. In *Proc. WAOA*, pages 51–68, 2018.
- 24 Marcin Bienkowski, Artur Kraska, and Paweł Schmidt. A match in time saves nine: Deterministic online matching with delays. In *Proc. WAOA*, pages 132–146, 2017.
- 25 Nadjib Brahimi, Stéphane Dauzere-Peres, Najib M Najid, and Atle Nordli. Single item lot sizing problems. *European Journal of Operational Research*, 168(1):1–16, 2006. doi:10.1016/J.EJOR.2004.01.054.
- 26 Carlos Fisch Brito, Elias Koutsoupias, and Shailesh Vaya. Competitive analysis of organization networks or multicast acknowledgment: How much to wait? *Algorithmica*, 64:584–605, 2012. doi:10.1007/S00453-011-9567-5.
- 27 Niv Buchbinder, Moran Feldman, Joseph Naor, and Ohad Talmon. O (depth)-competitive algorithm for online multi-level aggregation. In *Proc. SODA*, pages 1235–1244, 2017.
- 28 Niv Buchbinder, Tracy Kimbrel, Retsef Levi, Konstantin Makarychev, and Maxim Sviridenko. Online make-to-order joint replenishment model: primal dual competitive algorithms. In *Proc. SODA*, pages 952–961, 2008. URL: <http://dl.acm.org/citation.cfm?id=1347082.1347186>.
- 29 Maxim A. Bushuev, Alfred Guiffida, M.Y. Jaber, and Mehmood Khan. A review of inventory lot sizing review papers. *Management Research Review*, 38(3):283–298, 2015.
- 30 Ryder Chen, Jahanvi Khatkar, and Seeun William Umboh. Online weighted cardinality joint replenishment problem with delay. In *Proc. ICALP*, 2022.
- 31 Lindsey Deryckere and Seeun William Umboh. Online matching with set and concave delays. In *Proc. APPROX/RANDOM*, pages 17:1–17:17, 2023. doi:10.4230/LIPICS.APPROX/RANDOM.2023.17.
- 32 Daniel R. Dooley, Sally A. Goldman, and Stephen D. Scott. On-line analysis of the tcp acknowledgment delay problem. *Journal of the ACM*, 48(2):243–273, 2001. doi:10.1145/375827.375843.
- 33 Yuval Emek, Shay Kutten, and Roger Wattenhofer. Online matching: haste makes waste! In *Proc. STOC*, pages 333–344, 2016. doi:10.1145/2897518.2897557.
- 34 Yuval Emek, Yaacov Shapiro, and Yuyi Wang. Minimum cost perfect matching with delays for two sources. *Theoretical Computer Science*, 754:122–129, 2019. doi:10.1016/J.TCS.2018.07.004.
- 35 Leah Epstein. On bin packing with clustering and bin packing with delays. *Discrete Optimization*, 41:100647, 2021. doi:10.1016/J.DISOPT.2021.100647.
- 36 Leah Epstein. Open-end bin packing: new and old analysis approaches. *Discrete Applied Mathematics*, 321:220–239, 2022. doi:10.1016/J.DAM.2022.07.003.
- 37 Suresh K. Goyal and Ahmet T. Satir. Joint replenishment inventory control: deterministic and stochastic models. *European journal of operational research*, 38(1):2–13, 1989.
- 38 Anupam Gupta, Amit Kumar, and Debmalya Panigrahi. Caching with time windows. In *Proc. STOC*, pages 1125–1138, 2020. doi:10.1145/3357713.3384277.
- 39 Anupam Gupta, Amit Kumar, and Debmalya Panigrahi. A hitting set relaxation for k -server and an extension to time-windows. In *Proc. FOCS*, pages 504–515, 2022.

- 40 Sungjin Im, Benjamin Moseley, Chenyang Xu, and Ruilong Zhang. Online dynamic acknowledgement with learned predictions. In *Proc. INFOCOM*, pages 1–10, 2023. doi:10.1109/INFOCOM53939.2023.10228882.
- 41 Raf Jans and Zeger Degraeve. Modeling industrial lot sizing problems: a review. *International Journal of Production Research*, 46(6):1619–1643, 2008.
- 42 Dev Joneja. The joint replenishment problem: new heuristics and worst case performance bounds. *Operations Research*, 38(4):711–723, 1990. doi:10.1287/OPRE.38.4.711.
- 43 Naonori Kakimura and Tomohiro Nakayoshi. Deterministic primal-dual algorithms for on-line k-way matching with delays. In *Proc. ICCV*, pages 238–249, 2023. doi:10.1007/978-3-031-49193-1_18.
- 44 Behrooz Karimi, S.M.T. Fatemi Ghomi, and J.M. Wilson. The capacitated lot sizing problem: a review of models and algorithms. *Omega*, 31(5):365–378, 2003.
- 45 Anna R. Karlin, Claire Kenyon, and Dana Randall. Dynamic tcp acknowledgement and other stories about $e/(e-1)$. In *Proc. STOC*, pages 502–509, 2001. doi:10.1145/380752.380845.
- 46 Yasushi Kawase and Tomohiro Nakayoshi. Online matching with delays and size-based costs. *arXiv preprint arXiv:2408.08658*, 2024. doi:10.48550/arXiv.2408.08658.
- 47 Sanjeev Khanna, Joseph Seffi Naor, and Dan Raz. Control message aggregation in group communication protocols. In *Proc. ICALP*, pages 135–146, 2002.
- 48 Moutaz Khouja and Suresh Goyal. A review of the joint replenishment problem literature: 1989–2005. *European journal of operational Research*, 186(1):1–16, 2008. doi:10.1016/J.EJOR.2007.03.007.
- 49 Tung-Wei Kuo. Online deterministic minimum cost bipartite matching with delays on a line. *arXiv preprint*, 2024. doi:10.48550/arXiv.2408.02526.
- 50 Ngoc Mai Le, Seeun William Umboh, and Ningyuan Xie. The power of clairvoyance for multi-level aggregation and set cover with delay. In *Proc. SODA*, pages 1594–1610, 2023. doi:10.1137/1.9781611977554.CH59.
- 51 Ka-Cheong Leung, Victor OK Li, and Daiqin Yang. An overview of packet reordering in transmission control protocol (tcp): problems, solutions, and challenges. *IEEE Transactions on Parallel and Distributed Systems*, 18(4):522–535, 2007. doi:10.1109/TPDS.2007.1011.
- 52 Retsef Levi, Robin Roundy, David Shmoys, and Maxim Sviridenko. A constant approximation algorithm for the one-warehouse multiretailer problem. *Management Science*, 54(4):763–776, 2008. doi:10.1287/MNSC.1070.0781.
- 53 Retsef Levi, Robin Roundy, and David B Shmoys. Primal-dual algorithms for deterministic inventory problems. In *Proc. STOC*, pages 353–362, 2004. doi:10.1145/1007352.1007410.
- 54 Retsef Levi and Maxim Sviridenko. Improved approximation algorithm for the one-warehouse multi-retailer problem. In *Proc. APPROX-RANDOM*, pages 188–199, 2006. doi:10.1007/11830924_19.
- 55 Xingwu Liu, Zhida Pan, Yuyi Wang, and Roger Wattenhofer. Impatient online matching. In *Proc. ISAAC*, volume 123, pages 62:1–62:12, 2018. doi:10.4230/LIPICIS.ISAAC.2018.62.
- 56 Mathieu Mari, Michał Pawłowski, Runtian Ren, and Piotr Sankowski. Online matching with delays and stochastic arrival times. In *Proc. AAMAS*, pages 976–984, 2023.
- 57 Mathieu Mari, Michał Pawłowski, Runtian Ren, and Piotr Sankowski. Online multi-level aggregation with delays and stochastic arrivals. *arXiv preprint arXiv:2404.09711*, 2024.
- 58 Jeremy McMahan. A d -competitive algorithm for the multilevel aggregation problem with deadlines. *arXiv preprint arXiv:2108.04422*, 2021. arXiv:2108.04422.
- 59 Darya Melnyk, Yuyi Wang, and Roger Wattenhofer. Online k-way matching with delays and the h-metric. *arXiv preprint arXiv:2109.06640*, 2021. arXiv:2109.06640.
- 60 Tim Nonner and Alexander Souza. Approximating the joint replenishment problem with deadlines. *Discrete Mathematics, Algorithms and Applications*, 1(02):153–173, 2009. doi:10.1142/S1793830909000130.
- 61 Daniel Quadt and Heinrich Kuhn. Capacitated lot-sizing with extensions: a review. *Operation Research*, 6(1):61–83, 2008. doi:10.1007/S10288-007-0057-1.

- 62 Sheldon M. Ross. *Stochastic processes*, volume 2. Wiley New York, 1996.
- 63 Steven S. Seiden. A guessing game and randomized online algorithms. In *Proc. STOC*, pages 592–601, 2000. doi:10.1145/335305.335385.
- 64 Sombat Sindhuchao, H. Edwin Romeijn, Elif Akçali, and Rein Boondiskulchok. An integrated inventory-routing system for multi-item joint replenishment with limited vehicle capacity. *Journal of Global Optimization*, 32:93–118, 2005. doi:10.1007/S10898-004-5908-0.
- 65 Noam Touitou. Nearly-tight lower bounds for set cover and network design with deadlines/delay. In *Proc. ISAAC*, pages 53:1–53:16, 2021. doi:10.4230/LIPICS.ISAAC.2021.53.
- 66 Noam Touitou. Frameworks for nonclairvoyant network design with deadlines or delay. In *Proc. ICALP*, pages 105:1–105:20, 2023. doi:10.4230/LIPICS.ICALP.2023.105.
- 67 Noam Touitou. Improved and deterministic online service with deadlines or delay. In *Proc. STOC*, pages 761–774, 2023. doi:10.1145/3564246.3585107.
- 68 Wei Yuan, Srikanth V. Krishnamurthy, and Satish K. Tripathi. Synchronization of multiple levels of data fusion in wireless sensor networks. In *Proc. GLOBECOM*, volume 1, pages 221–225, 2003. doi:10.1109/GLOCOM.2003.1258234.