Online TCP Acknowledgment Under General Delays

Sujoy Bhore
Department of Computer Science & Engineering, Indian Institute of Technology Bombay, India

Michał Pawłowski
Faculty of Mathematics, Informatics and Mechanics, University of Warsaw, Poland
Department of Computer, Control and Management Engineering,
Sapienza University of Rome, Italy
IDEAS NCBR, Warsaw, Poland

Seeun William Umboh
School of Computing and Information Systems, The University of Melbourne, Australia
ARC Training Centre in Optimisation Technologies, Integrated Methodologies, and Applications (OPTIMA), Melbourne, Australia

Abstract

In a seminal work, Dooly, Goldman, and Scott (STOC 1998; JACM 2001) introduced the classic *Online TCP Acknowledgment* problem. In this problem, a sequence of n packets arrives over time, and the objective is to minimize both the number of acknowledgments sent and the total delay experienced by the packets. They showed that a natural greedy algorithm, which acknowledges when the delay of pending packets equals the acknowledgment cost, is 2-competitive.

Online TCP Acknowledgment is the canonical online problem with delay, capturing the fundamental tradeoff between reducing service cost through batching and the delay incurred by pending requests. Prior work has largely focused on richer service-cost models, e.g., Joint Replenishment and Multi-Level Aggregation. However, other than the work of Albers and Bals (SODA 2003), which studies maximum delay and closely related objectives, not much is known about general delay costs beyond the sum of delay costs of requests.

In this work, we study Online TCP Acknowledgment under two generalized delay-cost models that we call *batch-aware* and *batch-oblivious*. In the batch-aware model, each batch incurs a delay cost that depends on the packet delays within that batch. For the max-over-batches objective, which generalizes Albers and Bals, we show that greedy remains 2-competitive for every monotone batch-delay function. For the sum-over-batches objective, the picture changes sharply: greedy is $\Omega(n)$-competitive, and the optimal deterministic competitive ratio is $\Theta(\log n)$. Our matching upper bound requires only the minimal assumption that the batch delay function is monotone.

In the batch-oblivious model, the delay cost is a function of the global packet-delay vector. We show that greedy is 2-competitive for continuous submodular delay costs, and more generally under a weaker zero-coordinate diminishing-marginals condition. This yields 2-competitive algorithms for ℓ_p norms, Top-k norms, and ordered norms. Using the submodular-norm approximation of Patton, Russo, and Singla, we also obtain an $O(\log n)$-competitive algorithm for arbitrary symmetric norms.

2012 ACM Subject Classification Theory of computation → Online algorithms

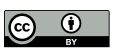
Keywords and phrases Online Algorithms, TCP Acknowledgment, General Delay Functions

Digital Object Identifier 10.4230/LIPIcs.APPROX/RANDOM.2026.26

Category APPROX

Related Version *Full Version*: https://arxiv.org/abs/2604.13428

Funding *Sujoy Bhore*: Work supported in part by ANRF ARG-MATRICS, Grant 002465.
Michał Pawłowski: Supported by the Polish National Science Center grant no 2024/53/N/ST6/04119.
Seeun William Umboh: Supported by the Australian Government through the Australian Research Council DP240101353.



1 Introduction

The *Online TCP Acknowledgment* problem, introduced by Dooly, Goldman, and Scott in their seminal STOC 1998 paper [28], arose from critical challenges in computer networking. Specifically, it addresses the dynamic management of acknowledgment delays within the Transmission Control Protocol (TCP), a cornerstone of the Internet's architecture (see [4, 24, 25]). The TCP protocol requires the recipient of a stream of packets to acknowledge every packet. On the other hand, excessive delay can disrupt TCP's congestion control mechanisms. Consequently, limiting the latency between a packet's arrival and its acknowledgment is vital for maintaining network performance and protocol stability.

In the Online TCP Acknowledgment problem – the TCP problem henceforth – a sequence of packets arrive online over time. At each time, we can transmit an acknowledgment to acknowledge all currently pending packets at a cost of 1. Each packet accrues a delay cost until it is acknowledged. More formally, the TCP problem involves a sequence of packet arrival times $A = (a_1, \ldots, a_n)$, and the objective is to partition A into k contiguous subsequences $\sigma_1, \sigma_2, \ldots, \sigma_k$, for some k. The end of each subsequence corresponds to the time an acknowledgment is transmitted. Thus, each σ_i is a *batch* of packets that are acknowledged together by the acknowledgment sent at time t_i. The batches σ_i and the acknowledgment times t_i are required to satisfy the following constraints: each packet can only be acknowledged after its arrival, i.e. $t_i \geq a_j$ for every $j \in \sigma_i$; and every packet must be acknowledged, i.e. the final acknowledgment time $t_k \geq a_n$, the arrival time of the last packet. The tradeoff between the cost of acknowledgments and the cost of delaying acknowledgments is modeled with the objective function

$$k + \sum_{i=1}^{k} \text{delay}(\sigma_i, t_i). \tag{1}$$

The number of acknowledgments k is called the *acknowledgment cost* and $\sum_{i=1}^{k} \text{delay}(\sigma_i, t_i)$ the *delay cost*. We refer to $\text{delay}(\cdot, \cdot)$ as the *batch delay cost function*.

TCP with Additive Delay Cost. The most classic and well-studied setting is when the delay function $\text{delay}(\sigma_i, t_i) = \sum_{j \in \sigma_i} (t_i - a_j)$, i.e. the delay cost is simply the total waiting time between each packet's arrival time and its acknowledgment time. The competitiveness of this problem has been completely characterized. For deterministic algorithms, Dooly, Goldman, and Scott [28, 29] proved that the following natural (in hindsight) Greedy algorithm achieves a competitive ratio of 2: acknowledge whenever the delay cost has increased by the cost of an acknowledgment. This is also the optimal deterministic competitive ratio as TCP with additive delay generalizes the well-known Ski Rental problem for which a lower bound of 2 is known [40]. Subsequently, Seiden [48] established a lower bound of $e/(e-1) \approx 1.58$ on the competitive ratio of randomized algorithms; a matching upper bound was shown by Karlin, Kenyon, and Randall [39] shortly after. These algorithms also work for slightly more general delay costs: each request j incurs a delay cost c_j that is a non-decreasing function of its waiting time, and the overall delay cost is $\sum_{j=1}^{n} c_j$. We call this more general problem *TCP with additive delay*, since the delay cost is the sum of the delay costs of the requests.

Beyond TCP. TCP with additive delay is the simplest problem in the class of *online problems with delay*, which captures trade-offs between batching requests to reduce service costs and minimize delays. An online problem with delay involves a sequence of requests that arrive online and a *service cost function* g which is a function on the set of requests; the

goal is to partition the requests into (possibly non-contiguous) batches and a service time t_i for each batch σ_i so as to minimize the objective function $\sum_i g(\sigma_i) + \text{delay}(\sigma_i, t_i)$. The sum $\sum_i g(\sigma_i)$ is called the *service cost*.

Most prior work on online problems with delay has focused on objective functions with more general service costs, with the overall delay cost computed as the sum of the delay costs of requests. These objective functions capture various types of economies of scale arising in varied practical domains such as scheduling, supply chain management, and logistics. The ones most relevant to our work are: joint replenishment [21, 16, 23, 37, 45, 49, 9, 27], multi-level aggregation [19, 20, 14, 15, 42], network design with delay [11, 12, 51], set cover [7, 50, 33], set aggregation [22], online service with delay [8, 18, 41, 35, 36, 52, 53], and matching [31, 6, 5, 17, 32, 10, 26, 38, 43, 30, 34].

Beyond Additive Delay Cost. In many of the above practical domains, it is natural to consider non-additive delay costs, where the penalty incurred by delaying requests is not simply the sum of their waiting times or delay costs. For example, one may wish to control tail latency by ensuring that no request waits too long, or, more generally, by penalizing the largest few waiting times. In settings with service-level agreements, another natural objective is to limit the number of requests that experience positive delay or exceed a prescribed delay threshold. Such objectives capture global quality-of-service or incident-based costs that are not well represented by additive delays.

The main goal of our work is to initiate a systematic study of online problems with delay beyond additive delay costs, starting with the setting that has the simplest service cost function: online TCP Acknowledgment. We investigate the following broad questions.

▶ **Question 1.** *For what families of natural delay costs can we obtain a good competitive ratio for the online TCP acknowledgment problem?*

As Greedy is heavily used as a subroutine in many algorithms for online problems with delay to determine service times (e.g. in [12, 23]), a good understanding of the power of Greedy can help us extend those algorithms to work with non-additive delay costs.

▶ **Question 2.** *For what families of natural delay costs is Greedy a good algorithm for the online TCP Acknowledgment problem?*

To the best of our knowledge, the only previous works on online problems with delay that go beyond additive delay costs are those below.

General Batch Delay Cost. Dooly, Goldman, and Scott [28, 29] considered a far-reaching generalization of the objective in Equation (1) where the batch delay cost function $\text{delay}(\cdot, \cdot)$ is only assumed to be monotone non-decreasing: if σ is a subsequence of σ' and $t \leq t'$, then $\text{delay}(\sigma, t) \leq \text{delay}(\sigma', t)$ and $\text{delay}(\sigma, t) \leq \text{delay}(\sigma, t')$. Equivalently, the marginal delay cost at each timestep depends only on the set of currently pending requests. These general batch-delay cost functions can be used to model not only the goal of avoiding requests waiting too long but also the goal of avoiding too many pending requests at any time. The former can be modeled using ℓ_p-norms, while the latter can be modeled by taking $\text{delay}(\sigma_i, t_i)$ to be a function of $|\sigma_i|$.[1] Dooly, Goldman, and Scott showed that Greedy is 2-competitive when $\text{delay}(\sigma_i, t_i) = \max_{j \in \sigma_i}(t_i - a_i)$, i.e. the batch delay cost is the maximum waiting time of

[1] This type of delay cost function was also studied under the name "size-based delay" by Deryckere and Umboh [26] and Gan, Sun, and Umboh [34] for online matching with delay.

the batch. They also claimed that Greedy is 2-competitive for monotone, nondecreasing batch-delay costs. However, we show that Greedy is actually $\Omega(n)$-competitive, and thus new algorithms are required.

Penalizing Long Delays. Later, Albers and Bals [2, 3] explored a structurally different objective function of the form

$$k + \max_{i=1}^{k} \max_{j \in \sigma_i} (t_i - a_j)^p$$

for some integer $p \geq 1$. When $p = 1$, the delay cost is simply the maximum waiting time of any packet. They gave tight upper and lower bounds on the competitive ratio of deterministic online algorithms: when $p = 1$, it is $\pi^2/6 \approx 1.644$; when $p > 1$, it is an expression that approaches 1.5 as p tends to infinity. For randomized algorithms, they showed lower bounds of $3/(3 - 2/e) \approx 1.324$ and $2/(2 - 1/e) \approx 1.225$, when $p = 1$ and when $p > 1$, respectively. A comprehensive overview of this area until 2003 can be found in Chapter 7.2 of the survey by Albers [1].

1.1 Our Contributions

In this work, we make significant progress towards answering the above questions. We consider three broad classes of delay costs that generalize those considered in previous works: *max-monotone*, *sum-monotone*, and *batch-oblivious*.

Max-Monotone Delay Cost. We begin by considering a generalization of the delay cost considered by Albers and Bals [3]. In the *max-monotone delay* setting, the objective function is

$$k + \max_{i=1}^{k} \text{delay}(\sigma_i, t_i) \tag{Max-Monotone}$$

for some monotone non-decreasing $\text{delay}(\cdot, \cdot)$. We show that Greedy is 2-competitive without any further restrictions on $\text{delay}(\cdot, \cdot)$ using a similar analysis as Greedy under additive delay costs.

▶ **Theorem 1.1.** *Greedy is a deterministic* 2*-competitive algorithm for TCP Acknowledgment with max-monotone delay cost.*

Sum-Monotone Delay Cost. Our main technical contributions are for the *sum-monotone delay* setting, where the objective is

$$k + \sum_{i=1}^{k} \text{delay}(\sigma_i, t_i) \tag{Sum-Monotone}$$

for some monotone non-decreasing $\text{delay}(\cdot, \cdot)$.

Intuitively, given the success of Greedy for max-monotone delay, and the apparent simplicity of the problem – that at any point in time, we only need to decide between two actions (to acknowledge or not) – one expects that Greedy is also good for sum-monotone. Indeed, Dooly, Goldman, and Scott [29, Theorem 10] claimed that the same analysis of Greedy under additive delays can be extended to sum-monotone delays. Surprisingly, we show that Greedy is actually terrible.

▶ **Theorem 1.2** (Informal; see Theorem 4.1). *Greedy is $\Omega(n)$-competitive for TCP Acknowledgment with sum-monotone delay cost.*

A further surprise is that there is no deterministic constant-competitive algorithm for sum-monotone delays. We give a lower bound of $\Omega(\log n)$ on the deterministic competitive ratio for sum-monotone delays via the Parking Permit problem – a generalization of Ski Rental – and a matching upper bound.

▶ **Theorem 1.3.** *The deterministic competitive ratio for TCP Acknowledgment with sum-monotone delay cost is $\Theta(\log n)$.*

Note that we measure the waiting time of a packet $j \in \sigma_i$ as $t_i - a_j$. However, all of our results also apply if the waiting time is transformed by any monotone, nondecreasing (possibly nonlinear) function before aggregation. Since all delay cost functions we consider are monotone non-decreasing, such transformations can equivalently be absorbed into the definition of the batch delay cost function $\text{delay}(\cdot, \cdot)$.

Batch-Oblivious Delay Cost. In the *batch-oblivious delay* setting, the delay cost is given by applying a monotone non-decreasing function f to the *delay vector* d of waiting times of each packet. More precisely, the j-th coordinate of the vector d is the waiting time of the j-th packet. To summarize, the batch-oblivious objective is

$$k + f(d). \tag{Batch-Oblivious}$$

Unlike max-monotone and sum-monotone delays, batch-oblivious delays are "global" objectives. For example, we can measure the ℓ largest waiting times among all requests by taking f to be the Top-ℓ norm;[2] note that this objective is not captured by sum-monotone and max-monotone delays.

In this setting, we show that a natural extension of Greedy is 2-competitive not only for ℓ_p norms and Top-ℓ norms, but also for a broader class of functions that are *continuous submodular*. The proof is much subtler than for additive delay.

▶ **Theorem 1.4.** *Greedy is a deterministic 2-competitive algorithm for TCP Acknowledgment with batch-oblivious continuous submodular delay cost.*

Continuous submodularity (Definition 5.1) generalizes the notion of submodularity for discrete functions to functions on real vectors [13]. Patton, Russo, and Singla [47] observed that ℓ_p norms and several other symmetric norms are continuous submodular. They also showed that any symmetric norm can be approximated by a continuous submodular norm up to a logarithmic factor. Using the results of [47], we get the following result.

▶ **Corollary 1.5.** *There is a deterministic 2-competitive algorithm for TCP Acknowledgment, where the delay cost is: an ℓ_p norm, a Top-ℓ norm, or an ordered norm. When the delay cost is a symmetric norm $\|\cdot\|$, then there is a deterministic $O(\log \rho)$-competitive algorithm where $\rho = \|(1, 1, \ldots, 1)\| \,/\, \|(1, 0, \ldots, 0)\| \leq n$. In particular, there is a deterministic $O(\log n)$-competitive algorithm when the delay cost is a monotone symmetric norm.*

Our final result shows that if we venture too far from norms to concave functions, then the problem becomes significantly more difficult.[3]

▶ **Theorem 1.6.** *Every deterministic algorithm is $\Omega(\sqrt{n})$-competitive for TCP Acknowledgment with batch-oblivious concave delay cost.*

[2] The Top-ℓ norm of a vector is the sum of its ℓ-largest entries.
[3] Recall that norms are convex.

1.2 Our Techniques

We first develop intuition for our algorithms by recalling the Greedy algorithm and its analysis for the classic additive delay costs. Then, we describe the modifications required to prove that Greedy is 2-competitive for max-monotone and continuous submodular delay costs. Finally, we give an overview of our algorithmic framework for sum-monotone delays, where greedy strategies are no longer competitive, and outline the phase-based approach that overcomes this barrier, together with the accompanying logarithmic lower bound.

Recap: Additive Delay. The Greedy algorithm can be easily described as follows: send an acknowledgment when the total increase in its delay cost since the last acknowledgment equals 1 – the cost of an acknowledgment. For additive delays, this is equivalent to sending an acknowledgment when the total waiting time of unacknowledged requests since the last acknowledgment reaches 1. Let us now analyze the competitive ratio of Greedy for additive delays. Suppose Greedy makes k acknowledgments at times $t_1 < \ldots < t_k$, with batches $\sigma_1, \ldots, \sigma_k$. The analysis consists of two main components:

1. By design, Greedy's acknowledgment cost equals its delay cost, so its cost is exactly $2k$.
2. The optimal solution pays at least 1 during each of the k intervals $I_j = [t_{j-1}, t_j)$ for $j \in [k]$, where $t_0 = 0$. This is because with additive delays, if the optimal solution does not send an acknowledgment within the time interval I_j, then the packets in batch σ_i contribute $\text{delay}(\sigma_i, t)$ to its delay cost.

While the first component still holds for any monotone delay cost, the second component breaks for the delay costs that we consider.

Max-Monotone Delay. For max-monotone delays, we get that $\text{delay}(\sigma_j, t_j) = j$ and if the optimal solution does not send an acknowledgment during interval I_j, then the packets of σ_j were acknowledged together at some time $t_j^* \geq t_j$, and so by monotonicity of $\text{delay}(\cdot, \cdot)$, the delay cost of the optimal solution is at least j. Thus, if the optimal solution makes $k^* < k$ acknowledgments, its delay cost is at least $k - k^*$, and so its total cost is at least k.

Sum-Monotone Delay. For sum-monotone delay costs, greedy strategies are not competitive. We show that for any constant threshold, there exists an instance where a greedy algorithm incurs a cost that is $\Omega(n)$ times larger than that of the offline optimum. This fundamental limitation stems from the additive nature of the delay cost: greedy algorithms react too locally, failing to recognize when it would be better to wait and serve a larger group of packets with a single acknowledgment. For instance, if the delay function is concave and grows steadily up to the acknowledgment cost, and if the packets are spaced far enough apart, greedy will serve each packet individually – incurring an acknowledgment cost each time – while the optimal solution serves all at once at the last packet arrival, paying the delay cost only once and saving on acknowledgments.

To overcome this, we use our offline dynamic programming solution to guide online decisions. The key idea is to monitor the cost of the optimal solution on the packets seen so far. We partition the execution of the algorithm into consecutive *services*, each of which aggregates packets and ends with an acknowledgment. At the start of a service, we examine the sequence of packets observed so far and identify the longest suffix for which the offline optimum issues a single acknowledgment. The cost of optimally serving this suffix determines a *budget* for the service. The algorithm then continues to accumulate packets and issues an acknowledgment as soon as the total delay cost exceeds this budget. Thus, the service budget is derived directly from the structure of an optimal offline solution on the currently observed packets.

To avoid repeatedly issuing acknowledgments with small accumulated delay costs, each budget service is followed by three *buffer services* whose budgets are larger by a constant factor than that of the preceding budget service. These buffer services separate consecutive budget services and later allow us to analyze their costs independently. Throughout the execution, the algorithm continues to monitor newly observed suffixes. During a budget service, the arrival of any longer critical suffix triggers an update of the service budget, whereas during a buffer service, such an update occurs only if the offline cost of a suffix increases by a constant factor. This controlled budget growth ensures that the number of services grows only logarithmically, implying that the total cost incurred by the algorithm is within an $O(\log n)$ factor of the offline optimum.

Lower Bound for Sum-Monotone Delay. Observe that if the batch delay cost depends only on the maximum waiting time of the requests in the batch, then the offline problem essentially reduces to an interval covering problem, where we seek to cover the arrival times with intervals whose cost is a function of the interval length. Thus, it makes sense to consider the Parking Permit Problem [44], a generalization of the classic Ski Rental problem. In this problem, we are given different permit types and for each permit type k, its cost C_k and duration D_k. Purchasing permit k at time t covers the time up to $t + D_k$. Online, a sequence of requests over time, and for each requested time t, we need to ensure that it is covered by a purchased permit. A deterministic lower bound of $\Omega(\log n)$ is known for this problem [46].

Offline, it is straightforward to reduce Parking Permit to our problem, since both problems are interval covering problems in which the cost of an interval depends only on its length. However, it is unclear how to do the reduction online. Intuitively, we would like to use an algorithm for our problem to solve Parking Permit as follows: when a request arrives for Parking Permit at time t, we give the same request to the TCP Acknowledgment algorithm, see how long it waits before acknowledging the request, and buy a permit of the corresponding length. Unfortunately, at the time the request arrives, the TCP Acknowledgment algorithm does not need to commit to when it will acknowledge the request. Indeed, if the next request arrives immediately after, it may be acknowledged at a different time than if it arrives much later. We leave the possibility of an online reduction from Parking Permit to our problem as an open, tantalizing problem.

Fortunately, the deterministic adversarial sequence for Parking Permit ensures that the next requested time is later than all the permits the algorithm has purchased so far. Thus, when the Parking Permit request arrives at time t, we pass it to the TCP Acknowledgment algorithm and simulate it into the future to determine at what time t^* it would acknowledge the request, assuming no further requests, and purchase the corresponding permit that covers up to time t^*. Since the next Parking Permit request arrives after the permit expires, the next TCP Acknowledgment request also arrives after t^*. Since the TCP Acknowledgment algorithm is online and the actual request sequence up to time t^* matches the simulation, it also acknowledges at time t^*. Thus, the TCP Acknowledgment algorithm and the Parking Permit algorithm behave identically, yielding the desired lower bound.

Continuous Submodular Delay. The proof of Theorem 1.4 requires a more subtle analysis of Greedy. The second part of the greedy analysis for additive delays breaks down even when the continuous submodular delay cost is a concave function of the total waiting time. Intuitively, this is because the optimal solution can "front-load" its delay cost by acknowledging, e.g., the first few packets a long time after their arrival. This increases the delay cost, so later packets contribute only a small marginal amount to it. To address this, we consider a hypothetical

solution that produces the same acknowledgments as the optimal solution, but each packet's waiting time is capped at its waiting time in Greedy. Since f is monotone, capping the waiting times can only decrease the delay cost. Thus, the cost of the hypothetical solution is at most the cost of the optimal solution, so it suffices to prove a lower bound on the cost of the hypothetical solution. The capping prevents the hypothetical solution from front-loading its delay cost and enables us to argue that if it does not make an acknowledgment during interval I_j, then its delay cost increases by at least 1 during the interval.

2 Preliminaries

Problem Setup. We consider the *Online TCP Acknowledgment* problem. The input is a sequence of packet arrival times $A = (a_1, a_2, \ldots, a_n)$, where a_j is the time at which the j-th packet arrives, and we assume $a_1 \leq a_2 \leq \cdots \leq a_n$. An *online algorithm* must select acknowledgment times $t_1 < t_2 < \cdots < t_k$, such that every packet is acknowledged at or after arrival – that is, for each $1 \leq j \leq n$, there exists some $1 \leq i \leq k$ with $a_j \leq t_i$. Each acknowledgment t_i covers the batch $\sigma_i = \{\, j : t_{i-1} < a_j \leq t_i \,\}$, where we set $t_0 = 0$. These batches form a partition of the packet indices $\{1, \ldots, n\}$. Each packet $j \in \sigma_i$ experiences a *waiting time* (or *delay*) $d_j = t_i - a_j$, and we denote the full delay vector by $d = (d_1, \ldots, d_n)$. The goal is to minimize the total cost, which typically balances two competing objectives: reducing the number of acknowledgments and limiting the total incurred delay. The online nature of the problem is that, as time t increases, the algorithm is aware only of requests whose arrival time is at most t and can only make an acknowledgment at the current time t.

Batch-Aware TCP. The *Batch-Aware TCP* model defines the delay cost in terms of the individual acknowledgment batches. Let $\text{delay}(\sigma_i, t_i)$ denote the delay cost of acknowledging batch σ_i at time t_i. The total delay cost is then computed from the per-batch costs, thereby capturing richer dependencies between the acknowledgment structure and delay.

In this work, we focus on two important classes of batch-aware cost models. In the *sum-monotone delay* model, the total cost is given by $k + \sum_{i=1}^{k} \text{delay}(\sigma_i, t_i)$. Here, the function $\text{delay}(\cdot, \cdot)$ is required to be monotone non-decreasing: if one batch is a subsequence of another and the acknowledgment times are ordered accordingly, the delay cost cannot decrease. This model captures objectives in which the overall delay accumulates across batches and scales with the batch size or delay duration.

In the *max-monotone delay* model, the objective is $k + \max_{1 \leq i \leq k} \text{delay}(\sigma_i, t_i)$, where $\text{delay}(\cdot, \cdot)$ is any monotone non-decreasing function. Unlike the sum-based case, this model focuses on minimizing the worst per-batch delay contribution, which is natural in applications where fairness or tail performance is critical.

Batch-Oblivious TCP. In the *Batch-Oblivious TCP* model, the delay cost is defined globally over the entire delay vector d. Specifically, the delay cost is given by a function $f_n(d_1, \ldots, d_n)$, where $f_n : \mathbb{R}_+^n \to \mathbb{R}_+$ is assumed to be monotone and consistent under zero-padding. Since the number of packets n is not known in advance, we assume the algorithm learns the function f_i only when the ith packet arrives. These functions satisfy the consistency condition

$$f_{i+1}(d_1, \ldots, d_i, 0) = f_i(d_1, \ldots, d_i),$$

ensuring that appending a packet with zero delay does not affect the cost. A natural example of this model is total waiting time, where $f_n(d) = \sum_{j=1}^{n} d_j$. The overall objective in this setting is to minimize $k + f_n(d)$, where k denotes the number of acknowledgments. We refer to this setting as batch-oblivious because the delay cost treats all packet delays globally, regardless of which acknowledgment batch a packet belongs to.

3 Max-Monotone Delay Cost

We begin our investigation with the max-monotone delay model, where the total cost is given by the number of acknowledgments plus the maximum per-batch delay cost. We show that the natural greedy algorithm is 2-competitive.

▶ **Theorem 1.1.** *Greedy is a deterministic 2-competitive algorithm for TCP Acknowledgment with max-monotone delay cost.*

Proof. Recall that we assume the acknowledgment cost is normalized to 1. We analyze the natural greedy algorithm that sends an acknowledgment whenever the total cost increases by 1. Since we are in the max-monotone model, this is equivalent to acknowledging a batch of packets once its delay cost reaches the current batch index. For example, the algorithm sends the first acknowledgment when the currently pending packets accumulate delay cost 1, then the second when the pending batch reaches delay cost 2, and so on. In this way, each acknowledgment increases the total cost by exactly 1 from the delay, in addition to the unit cost of the acknowledgment.

Let t_j denote the time when the algorithm makes its j-th acknowledgment, and define the interval $T_j = (t_{j-1}, t_j]$, with $t_0 = 0$. Moreover, let σ_j be the batch of packets acknowledged at time t_j. By design, we have $\text{delay}(\sigma_j, t_j) = j$, since the algorithm waited until this batch's delay cost reached exactly j. Thus, the total cost of the algorithm is k from acknowledgments and $\max_{1 \le j \le k} \text{delay}(\sigma_j, t_j) = k$ from delay, totaling at most $2k$.

Denote by k^* the number of acknowledgments made by the optimal solution. We now argue that its total cost is at least k, which implies that the greedy algorithm is 2-competitive.

▷ Claim 3.1. If the optimal solution does not send an acknowledgment during interval T_j, then its delay cost is at least j.

Proof. Since no acknowledgment is made during T_j, all packets in σ_j must be acknowledged by the optimal solution in a later batch σ^*, sent at time $t^* \ge t_j$. Because $\text{delay}(\cdot, \cdot)$ is monotone non-decreasing in both batch size and acknowledgment time, it follows that $\text{delay}(\sigma^*, t^*) \ge \text{delay}(\sigma_j, t_j)$, which equals j, proving the claim. ◁

Suppose that the optimal solution makes k^* acknowledgments. If $k^* \ge k$, then we are done. Otherwise, Claim 3.1 implies that the delay cost of the optimal solution is at least $k - k^*$ (as the acknowledgments can be placed in the last k^* T_i intervals) and so the optimal solution has total cost at least k, as desired. ◀

4 Sum-Monotone Delay Cost

In this section, we focus on TCP acknowledgment under the sum-monotone delay-cost model. First, we prove that any deterministic greedy policy is $\Omega(n)$-competitive: there exists an input of size n that enforces the greedy algorithm to pay n-fold more than the offline optimum. We then turn to the offline problem and present a polynomial-time algorithm via dynamic programming, that computes an exact optimum schedule. By embedding this offline oracle into an online framework, we obtain a new algorithm with an $O(\log n)$ competitive ratio: at each decision point, it computes the offline optimum schedule for the set of all packets that have arrived so far and uses that schedule to guide its acknowledgments. Finally, we show that every deterministic algorithm is $\Omega(\log n)$-competitive.

4.1 Lower Bound for Greedy

We begin with a more precise statement of Theorem 1.2. Elsewhere in this paper, when we talk about the Greedy algorithm, we refer to the algorithm that sends an acknowledgment whenever $\text{delay}(\sigma, t) = 1$, where σ is the set of currently unacknowledged packets and t is the current time. Our lower bound holds for a family of greedy algorithms: for $\tau > 0$, algorithm Greedy_τ sends an acknowledgment when $\text{delay}(\sigma, t) = \tau$.

▶ **Theorem 4.1.** *For every $\tau > 0$, the algorithm Greedy_τ is $\Omega(n)$-competitive for TCP Acknowledgment with sum-monotone delay cost.*

Proof. Fix $\tau > 0$. Define the batch delay function $\text{delay}(\sigma, t) = \min\{\sum_{i\in\sigma} d_i, \tau\}$, where $d_i = a_i - t$. Consider the sequence of n packet with arrival times $a_i = i(\tau + \epsilon)$ for a small enough ϵ. Observe that Greedy_τ makes an acknowledgment exactly τ time steps after the arrival of each packet. Thus, Greedy_τ makes n acknowledgments and incurs a delay cost of $n\tau$ for a total cost $n(1 + \tau)$. On the other hand, the solution that waits to make a single acknowledgment at the arrival time of the last packet a_n has a total cost of at most $1 + \tau$. Thus, the competitive ratio of Greedy_τ is at least n. ◀

4.2 Offline Model

Solving the offline version of the TCP acknowledgment under the sum-monotone delay-cost model can be efficiently performed using dynamic programming. The general structure of the solution closely follows the approach used by Dooly et al. [29], though here we consider a general batch cost function $\text{delay}(\sigma, t)$, making our model a natural generalization of their framework.

Let σ be the given sequence of packets, and $\sigma|_i^j$ be its subsequence containing packets i through j. We define the subproblems for our dynamic programming solution as follows. Let $\text{DP}[i]$ denote the minimum total cost of acknowledging packets 1 through i, and set $\text{DP}[0] = 0$. Then, the optimal cost can be computed using the following recurrence relation:

$$\text{DP}[i] = \min_{1\le j\le i} \left\{\text{DP}[j-1] + \text{delay}\left(\sigma|_j^i, a_i\right) + 1\right\}.$$

Intuitively, at each step i, the algorithm evaluates all possible previous acknowledgment positions j and selects the one that minimizes the sum of the optimal cost up to $j - 1$, the delay cost $\text{delay}(\sigma|_j^i, a_i)$, and the acknowledgment cost 1.

The correctness of this approach follows directly from the optimal substructure property: an optimal partitioning of packets into acknowledged subsequences necessarily implies that each of these subsequences itself must be an optimal solution to the corresponding subproblems. The computational complexity of this method is $O(n^2)$, since evaluating each $\text{DP}[i]$ involves examining $O(n)$ previous acknowledgment points, and we have n such subproblems.

4.3 Online Model

Using the offline result from the previous subsection, we now prove the following lemma.

▶ **Lemma 4.2.** *There is a deterministic $O(\log n)$-competitive algorithm for TCP Acknowledgment with sum-monotone delay cost.*

The key idea is to use the cost of the optimal offline solution on the sequence of packets seen so far to guide when our algorithm issues acknowledgments. Suppose the j-th packet arrives and there are no pending packets; we then start a new *budget service s*. Let $\sigma|^j$ denote

the sequence of the first j packets observed so far.[4] We first identify the longest *critical suffix* of $\sigma|^j$, that is, a suffix $\sigma_s \subseteq \sigma|^j$ such that the optimal offline strategy for σ_s consists of issuing a single acknowledgment exactly at the arrival time of its last packet. Observe that at least one such suffix always exists, namely the one containing only packet j. Moreover, notice that for a critical suffix σ_s, the optimal cost of serving it equals $\text{delay}(\sigma_s, a_j) + 1$. To simplify notation, we define $\text{serve}(\sigma_s, a_j) = \text{delay}(\sigma_s, a_j) + 1$. Whenever an acknowledgment is issued at the arrival time of the last packet in the suffix, we omit the second argument and write $\text{serve}(\sigma_s)$.

Based on σ_s, we define a budget b_s for service s (its precise definition will be given later). The service s then aggregates arriving packets and issues an acknowledgment as soon as their total accumulated delay cost reaches b_s; we refer to this moment as the *critical time* of service s. If, during the aggregation phase of a budget service, a new packet arrives, we again compute the longest critical suffix $\sigma' \subseteq \sigma|^j$. If σ' strictly extends σ_s (that is, if it starts earlier than σ_s), we update the budget b_s accordingly to reflect the increased cost suggested by the offline optimum.

When a budget service s finally issues its acknowledgment, the algorithm schedules three consecutive *buffer services*, each with a budget equal to twice the budget of s. Each buffer service aggregates packets and acknowledges them as soon as their delay cost reaches its assigned budget, and this process is repeated three times.[5] During a buffer service, if a packet arrives that induces a critical suffix whose cost is at least twice the cost of σ_s, then the current service is immediately promoted to a new budget service, with its budget defined accordingly.

Assigning exponentially larger budgets to buffer services prevents the algorithm from issuing many low-cost acknowledgments in situations where the offline optimum would issue a single high-cost acknowledgment. By scaling service budgets exponentially in response to increasing costs of critical suffixes, the algorithm ensures that its total cost increases only logarithmically relative to the optimal offline cost, yielding an $O(\log n)$ competitive ratio.

In what follows, we use $\texttt{OPT}(\sigma_s)$ to denote the cost of the optimal offline solution for batch σ_s.

4.3.1 Algorithm Description

We begin by formalizing the notion of *critical batches*, which correspond to the critical suffixes discussed earlier and will play a central role in the design and analysis of our algorithm.

▶ **Definition 4.3** (Critical Batch). *We call batch* σ critical *if it satisfies* $\texttt{OPT}(\sigma) = \text{serve}(\sigma)$.

We are now ready to present Algorithm 1. This algorithm is structured around two event-driven functions: "UponArrival", which is triggered whenever a new packet arrives, and "UponCriticalTime", which is triggered when a precomputed critical time is reached. Together, these functions manage the scheduling of acknowledgments based on the observed input structure.

When a packet arrives, the `UponArrival` appends it to the already-seen packets sequence and finds the largest critical suffix. Then, depending on whether there is a service running and if so – of what type it is – we possibly update the budget of the service (and its type, as in line 11). Notice that line 13 sets the critical time t_c so that acknowledging all currently

[4] Here we slightly overload notation: earlier, $\sigma|^j$ denoted the j-th batch served by the algorithm.
[5] The purpose of these buffer services is to separate consecutive budget services and simplify the analysis.

Algorithm 1 Sum-Monotone TCP Algorithm.

Initialization: sequence of already seen packets $\sigma = []$, critical time $t_c = 0$, critical batch $\sigma_c = []$ and budget b

Function UponArrival(r):

1: put r at the end of σ
2: let σ' be the largest critical suffix
3: **if** there is no service running, i.e., σ_c is empty **then** ▷ start a buffer service
4: $\sigma_c = \sigma'$
5: **else if** there is a budget service running **then**
6: **if** $\sigma_c \subseteq \sigma'$ **then** ▷ check if the current critical suffix is a larger
7: $\sigma_c = \sigma'$ ▷ if so, update the critical batch
8: **else if** there is a buffer service running **then**
9: **if** $\text{serve}(\sigma') \geq 2 \cdot \text{serve}(\sigma_c)$ **then** ▷ check whether the critical suffix is large enough
10: $\sigma_c = \sigma'$ ▷ if so, update the critical batch
11: switch the current service to a budget service
12: $b = 2 \cdot \text{serve}(\sigma_c)$ ▷ update the critical batch
13: let t_c be such that $\text{serve}(\text{pending packets}, t_c) = b$ ▷ update the critical time

Function UponCriticalTime(): ▷ called when t_c is reached

1: make an acknowledgment
2: **if** the budget service ended **then**
3: start a new buffer service and double the budget $b \leftarrow 2b$
4: **else if** the first or second buffer service ended **then**
5: start a consecutive buffer service
6: **else if** the third buffer service ended **then**
7: $\sigma_c = []$, $b = 0$ ▷ reset the variables

pending packets at time t_c will incur a delay cost exactly equal to the budget b. If this cost never reaches b, we assume $t_c = \infty$, meaning the current service will conclude only at the end of the input sequence.

The `UponCriticalTime` function is called when t_c is reached. It performs an acknowledgment and issues a new service, or resets the state if all buffer services have completed.

4.3.2 Critical Batches and Relations Between Them

We begin by introducing some notation. Suppose that Algorithm 1 issues k acknowledgments for budget services over a sequence of n requests. We denote the corresponding critical batches by σ_c^i for $i \in \{1, 2, \ldots, k\}$. Each critical batch may be followed by up to three buffer services. We denote by $\sigma_{b,l}^i$, for $l \in \{1, 2, 3\}$, the sequences of packets acknowledged by these services. If a particular buffer service is not started, we define the corresponding batch to be the empty sequence.

▶ **Definition 4.4** (Critical Batch Intersection). *We say that the critical batch σ_c^j* intersects *σ_c^i, $i < j$, if*

(1) *$j = i + 1$ and the budget service corresponding to σ_c^j was started in line 11 of the `UponArrival` function in Algorithm 1, or*

(2) $\sigma^i_{b,2}$ *and* $\sigma^i_{b,3}$ *are both not empty, while* σ^j_c *and* $\sigma^i_{b,2}$ *share at least one packet, that is,* $\sigma^j_c \cap \sigma^i_{b,2} \neq \emptyset$.

When using set-theoretic notation, we implicitly identify a sequence with the set of packets it contains.

Understanding how critical batches intersect is key to analyzing the proposed algorithm. To make our reasoning simpler, we introduce a few new terms to describe this relation.

▶ **Definition 4.5** (Parent Batch). *Given a critical batch* σ^i_c*, we define the set* $\texttt{children}(\sigma^i_c)$ *to consist of all earlier critical batches that* σ^i_c *intersects and for which* σ^i_c *is the first subsequent critical batch to do so. For each batch* $\sigma^j_c \in \texttt{children}(\sigma^i_c)$*, we refer to* σ^i_c *as its* parent batch*. We use the terms* ancestor *and* descendant *to denote the transitive closure of this relation.*

▶ **Definition 4.6** (Final Critical Batch). *We call a batch with no parent a* final critical batch.

In what follows, we also use the following notation for the pivotal packet arrival times. We use t^i_s and t^i_c to denote the first and the last packet's arrival times in σ^i_c. Moreover, if σ^i_c is a parent batch, we let σ^i_e be its descendant that was acknowledged the earliest, and let t^i_e be the arrival of the first packet in σ^i_e.

▶ **Definition 4.7** (Coverage Interval). *Given a critical batch* σ^i_c*, we define its* coverage interval *as follows. If* σ^i_c *has no children, then its coverage interval is* $[t^i_s, t^i_b]$*; otherwise, it is* $[t^i_e, t^i_b]$*. We denote this interval by* I^i_c*. When convenient, we also use* I^i_c *to denote the set of packets arriving within this time interval.*

4.3.3 Analysis

Let us start by showing how final critical batches relate to the optimal cost of serving packets.

▶ **Lemma 4.8.** *Let* $\sigma^1_c, \ldots, \sigma^\ell_c$ *be a sequence of final critical batches obtained during a run of Algorithm 1 on a packet sequence* $\hat{\sigma}$*. We assume that they are listed in the order in which the corresponding budget services were issued, and that the list contains all such batches. Then, it holds that*

$$\texttt{OPT}\left(\hat{\sigma}\right) \geq \sum_{i=1}^{\ell} \texttt{OPT}\left(I^i_c\right).$$

Proof. Let $\hat{\sigma}_1, \ldots, \hat{\sigma}_p$ denote the partition of $\hat{\sigma}$ induced by the optimal solution, where each $\hat{\sigma}_i$ corresponds to a single acknowledgment. By definition, each $\hat{\sigma}_i$ forms a critical batch and therefore must be recognized by Algorithm 1.

Suppose that for some $j \in \{1, \ldots, p\}$, the batch $\hat{\sigma}_j$ overlaps with at least two coverage intervals corresponding to the final critical batches identified by the algorithm. Let σ^k_c, for some $k \in \{1, \ldots, \ell\}$, be the latest such critical batch, and let $\hat{j}$ denote the last packet in $\hat{\sigma}_j$. Finally, let $\sigma^{k,j}_c$ be the largest critical batch among σ^k_c and its descendants such that $\hat{j}$ belongs to one of the services associated with this batch.

If $\hat{j}$ is served by the budget service associated with $\sigma^{k,j}_c$, then at the moment $\hat{j}$ arrives, the `UponArrival` function would recognize $\hat{\sigma}_j$ in either line 4 or 7 and act upon it. Otherwise, if $\hat{j}$ is served by a buffer service, it would trigger the start of a new budget service in line 11. Indeed, since $\hat{\sigma}_j$ overlaps with all packets acknowledged by the budget service corresponding to $\sigma^{k,j}_c$, its service cost exceeds $2 \cdot \text{serve}(\sigma^{k,j}_c)$, as stated in line 12, and thus the condition in line 9 is satisfied.

Therefore, no batch served by the optimal solution can overlap with more than one coverage interval. Consequently, for each coverage interval I_c^i, we consider the set of all batches $\hat{\sigma}_j$ that overlap with it. This set covers at least as many packets as I_c^i, and hence its total cost is at least $\texttt{OPT}(I_c^i)$. Since each batch $\hat{\sigma}_j$ is counted at most once, the desired inequality follows. ◀

From there, we obtain two straightforward properties.

▶ **Corollary 4.9.** *The total cost of* OPT *on a given sequence of packets σ is at least equal to the sum of the optimal costs of serving coverage intervals of all the final critical batches recognized during the run of Algorithm 1 on σ.*

▶ **Corollary 4.10.** *For any critical batch with children, the optimal cost of serving its coverage interval is at least the sum of the children's critical interval optimal serving costs.*

Now, observe that also the following holds for any critical batch with children.

▶ **Lemma 4.11.** *Let σ_c^i be a critical batch such that* $\texttt{children}(\sigma_c^i) \neq \emptyset$*. Then, for any child batch σ_c^j, we have* $\text{serve}(\sigma_c^i) \geq 2 \cdot \text{serve}(\sigma_c^j)$.

Proof. Since σ_c^i is a parent of σ_c^j, Definition 4.5 implies that σ_c^i intersects σ_c^j. Therefore, one of the two conditions in Definition 4.4 – namely, 1 or 2 – must hold.

We first consider condition 1. In this case, we have $j = i + 1$, and the budget service corresponding to σ_c^j is initiated in line 11 of the `UponArrival` function by some critical suffix σ'. Consequently, the condition in line 9 holds, which implies that $\text{serve}(\sigma') \geq 2 \cdot \text{serve}(\sigma_c^i)$. Since $\sigma' \subseteq \sigma_c^j$ (as σ' may have been extended in line 6 to become σ_c^i), the claimed inequality follows.

We now turn to condition 2. In this case, both $\sigma_{b,2}^i$ and $\sigma_{b,3}^i$ are non-empty, and σ_c^j shares at least one packet with $\sigma_{b,2}^i$, that is, $\sigma_c^j \cap \sigma_{b,2}^i \neq \emptyset$. Since σ_c^j overlaps with $\sigma_{b,2}^i$, it must also cover $\sigma_{b,3}^i$, which immediately follows $\sigma_{b,2}^i$. As $\sigma_{b,3}^i \neq \emptyset$, its service cost satisfies $\text{serve}(\sigma_{b,3}^i) = 2 \cdot 2 \cdot \text{serve}(\sigma_c^i)$, by lines 3 of the `UponCriticalTime` function and 12 of the `UponArrival` function. Hence, $\text{serve}(\sigma_c^j) \geq \text{serve}(\sigma_{b,3}^i) = 4 \cdot \text{serve}(\sigma_c^i)$, and the desired inequality also holds in this case. ◀

▶ **Lemma 4.12.** *The budget service associated with the critical batch σ_c^i incurs a cost of $2 \cdot \text{serve}(\sigma_c^i)$, while each buffer service associated with σ_c^i incurs a cost of at most $4 \cdot \text{serve}(\sigma_c^i)$.*

Proof. By the definition of the algorithm. ◀

▶ **Lemma 4.13.** *Every final critical batch σ_c^i has at most $\log(\texttt{OPT}(I_c^i)) \leq \log n$ levels of descendant sets.*

Proof. By Lemma 4.11, each child batch has a cost at least twice that of the parent. Since the minimum cost is 1 (the acknowledgment cost), the first property follows. To observe the upper bound on this value, notice that one of the possible solutions is to acknowledge each packet individually, and thus $\texttt{OPT}(I_c^i) \leq n$. ◀

Combining the above lemmas gives us Lemma 4.2. Indeed, by Lemma 4.13 and a recursive application of Corollary 4.10, we have that the sum of optimal costs for all the critical batches processed by our algorithm (i.e., those for which a service was issued) is at most equal to $\log n \cdot \sum_{i=1}^{\ell} \texttt{OPT}(I_c^i)$. Thus, by Lemma 4.12, the algorithm paid at most $14 \log n \cdot \sum_{i=1}^{\ell} \texttt{OPT}(I_c^i)$ to serve all the packets. The lower bound on optimal cost from Corollary 4.9 completes the proof.

4.4 Lower Bounds for Arbitrary Algorithms

We show that the deterministic $O(\log n)$ competitive ratio of our algorithm (Algorithm 1) is tight. We start by defining the Parking Permit Problem [44]. It is known that for every deterministic algorithm $\mathcal{B}$ for Parking Permit, there is an adversarial request sequence A on which $\mathcal{B}$ is $\Omega(\log n)$-competitive (Lemma 4.14 [46]). Then, we show how to convert any deterministic algorithm $\mathcal{A}$ for TCP Acknowledgment into a deterministic algorithm $\mathcal{B}_{\mathcal{A}}$ for Parking Permit. Finally, we show that $\mathcal{A}$ is $\Omega(\log n)$-competitive on the adversarial sequence for $\mathcal{B}_{\mathcal{A}}$ (Lemma 4.17).

Parking Permit Problem [44]. An instance of the Parking Permit Problem. For each timestep $t \geq 1$, a request may arrive, and if so, needs to be covered by a permit. There are several different types of permits that can be purchased at any time throughout the time horizon. A permit of type $k \geq 0$ has cost C_k and duration D_k: when bought at time t, it covers the time interval $[t, t + D_k]$. The goal is to minimize the cost of the purchased permits.

Hard Parking Permit instances. A *hard Parking Permit instance* is one where $C_k = 2^k$ and $D_k = 4^k$ for each $k \geq 0$.

▶ **Lemma 4.14** ([46]). *Let $\mathcal{B}$ be a deterministic online algorithm for Parking Permit. Consider a hard Parking Permit instance where the sequence of request arrival times A is generated by the following adversary: for each $1 \leq j \leq n$ (in ascending order), request j arrives at the earliest timestep that is not yet covered by the permits bought by $\mathcal{B}$.*

When run on A, algorithm $\mathcal{B}$

- *buys a separate permit $[a_j, t_j]$ for each request j; moreover, $a_{j+1} = t_j + 1$ for each $j < n$;*
- *has competitive ratio $\Omega(\log n)$.*[6]

Hard TCP Acknowledgment instances. We now define the corresponding TCP Acknowledgment instances. Define the piecewise-linear concave function $f(x) = \min_{k \geq 0} C_k + x \cdot (C_k/D_k)$ where C_k and D_k are defined as above. A *hard TCP Acknowledgment instance* is one in which the total cost of a batch σ acknowledged at time t – i.e., sum of the acknowledgment cost[7] and the delay cost of the batch $\text{delay}(\sigma, t)$ – is given by $f(t - a(\sigma))$ where $a(\sigma)$ is the earliest arrival time among the packets in σ. That is, $\text{delay}(\sigma, t) = f(t - a(\sigma)) - 1$.

Converting TCP Acknowledgment algorithm to Parking Permit. Observe that a hard Parking Permit instance with arrival sequence A has a corresponding hard TCP Acknowledgment instance with the same arrival sequence. Our goal now is to show how to map decisions of a TCP Acknowledgment algorithm on the corresponding instance into decisions for the Parking Permit instance. We begin by presenting the two necessary ingredients.

First, we need to decide which permit to buy when a request arrives even though the TCP Acknowledgment algorithm is not required to commit on arrival of a request, when the request will be acknowledged. For a sequence A, let A_j denote the subsequence containing the first j elements. The following lemma, together with Proposition 4.16, lets us decide which permit to buy on arrival of a request.

[6] We remark that the Parking Permit Problem is typically defined as having a time horizon $[1, N]$ (on which n requests arrive) and Lemma 21 of [46] gives a lower bound of $\Omega(\log N)$. Since $N \geq n$, we get a lower bound of $\Omega(\log n)$.

[7] Recall that the acknowledgment cost is 1.

▶ **Lemma 4.15.** *Let $\mathcal{A}$ be a deterministic online algorithm for TCP Acknowledgment. Then, there exists a function* next *such that for every sequence of packet arrival times $A = (a_1, \ldots, a_n)$ and for every j, if $a_{j+1} > \text{next}(A_j)$, then packet j is acknowledged at time $\text{next}(A)$. The function* next *is called the* threshold function *of $\mathcal{A}$.*

Proof. Fix a sequence A and $1 \leq j \leq n$. Define $\text{next}(A_j) \geq a_j$ to be the time that $\mathcal{A}$ acknowledges the last packet of A_j when $\mathcal{A}$ is run on A_j. If $a_{j+1} > \text{next}(A_j)$, then up till time $\text{next}(A_j)$, the executions of $\mathcal{A}$ on A_j and of $\mathcal{A}$ on A see the same information. Since $\mathcal{A}$ is an online algorithm, the execution of $\mathcal{A}$ on A will also acknowledge at time $\text{next}(A_j)$. ◀

Second, we need to be able to map batches to permits. In general, the time between when a batch is acknowledged and the earliest arrival time of the batch can be arbitrary, while permits can only have lengths $D_k = 4^k$. The following lets us approximately map batches to permits.

▶ **Proposition 4.16.** *For every $x \geq 0$, there exists $k^* \geq 0$ such that $D_{k^*} \geq x$ and $C_{k^*} \leq 2f(x)$.*

Proof. Suppose $f(x) = C_k + x(C_k/D_k)$. If $x < D_k$, then $f(x) \geq C_k$ so choosing $k^* = k$ works. Next, we consider the case when $x > D_k$. Let $k^* > k$ be such that $D_{k^*-1} \leq x \leq D_{k^*}$. Observe that $C_k/D_k = (1/2)^k \leq (1/2)^{k^*-1} = C_{k^*-1}/D_{k^*-1}$. Together with the fact that $x \geq D_{k^*-1}$, we have

$$f(x) \geq x(C_k/D_k) \geq D_{k^*-1}(C_{k^*-1}/D_{k^*-1}) = C_{k^*-1} = C_{k^*}/2,$$

as desired. ◀

We now have the ingredients to convert any deterministic algorithm $\mathcal{A}$ for hard TCP Acknowledgment instances to a deterministic algorithm $\mathcal{B}_\mathcal{A}$ for hard Parking Permit instances. Let $\text{next}(\cdot)$ be the threshold function of $\mathcal{A}$. The algorithm $\mathcal{B}_\mathcal{A}$ works as follows. For each request arrival a_j that is not covered by permits bought so far:

(1) let A_j be the sequence of arrival times so far, including a_j;
(2) let k^* be such that $D_{k^*} \geq x$ and $f(D_{k^*}) \leq 2f(\text{next}(A_j) - a_j)$ as guaranteed by Proposition 4.16;
(3) buy permit $[a_j, a_j + D_{k^*}]$.

▶ **Lemma 4.17.** *Let $\mathcal{A}$ be a deterministic algorithm for hard TCP Acknowledgment instances and A^* be the adversarial sequence for $\mathcal{B}_\mathcal{A}$ given by Lemma 4.14. Then, $\mathcal{A}$ is $\Omega(\log n)$-competitive on A^*.*

Proof. Let $\texttt{OPT}_{\text{PP}}(A^*)$ and $\texttt{OPT}_{\text{TCP}}(A^*)$ be the cost of the optimal Parking Permit and TCP Acknowledgment solutions to A^*, respectively. Since Lemma 4.14 already shows that $\mathcal{B}_\mathcal{A}$ is $\Omega(\log n)$-competitive on A^*, it suffices to show the cost of $\mathcal{A}$ on A^* is at least $\Omega(1)$ times the cost of $\mathcal{B}_\mathcal{A}$ on A^*, and that $\texttt{OPT}_{\text{TCP}}(A^*) = O(1)\texttt{OPT}_{\text{PP}}(A^*)$.

By Lemma 4.14 and by definition of $\mathcal{B}_\mathcal{A}$, for every $j < n$, we have that $a_{j+1} > a_j + D_{k^*} \geq \text{next}(A_j)$. Thus, for each request j, Lemma 4.15 implies that $\mathcal{A}$ acknowledges j by itself at time $\text{next}(A_j)$ and the total cost of the batch is $f(\text{next}(A_j) - a_j))$; on the other hand, $\mathcal{B}_\mathcal{A}$ buys permit $[a_j, a_j + D_{k^*}]$ of cost $C_{k^*} \leq 2f(\text{next}(A_j) - a_j)$. Thus, the cost of $\mathcal{A}$ on A^* is at least $1/2$ the cost of $\mathcal{B}_\mathcal{A}$ on A^*.

Next, we show that $\texttt{OPT}_{\text{TCP}}(A^*) = O(1)\texttt{OPT}_{\text{PP}}(A^*)$. Let $P_i = [s_i, t_i]$ be the i-th permit bought by the optimal solution for the hard instance of Parking Permit with arrival sequence A. We construct a solution to the hard TCP Acknowledgment instance as follows: while there is an unacknowledged packet, let a_j be the earliest unacknowledged packet and send an acknowledgment at time t where t is the furthest end time among all permits covering a_j (at least one such permit exists since the permits have to cover A).

We now argue that we can charge the total cost of each batch to a unique permit. For each batch σ_i acknowledged at time t_i, let $P_{(i)}$ be the permit that covers $a(\sigma_i)$ and has end time t_i. Since $P_{(i)}$ is the permit with the furthest end time, we have that $P_{(i')} \neq P_{(i)}$ for $i' \neq i$. Let $C_{(i)}$ and $D_{(i)}$ be the cost and duration of $P_{(i)}$. Since $a(\sigma_i) \in P_{(i)}$, we get that the total cost of σ_i is $f(t_i - a(\sigma_i)) \leq f(D_{(i)}) = 2C_{(i)}$. Thus the cost of each batch σ_i is charged to at most twice the cost of permit $P_{(i)}$, and each permit is charged by at most one batch. Thus, we have shown that $\texttt{OPT}_{\text{TCP}}(A^*) = O(1)\texttt{OPT}_{\text{PP}}(A^*)$, completing the proof of the lemma. ◀

Thus, we conclude that every deterministic algorithm for TCP Acknowledgment with sum-monotone delay costs is $\Omega(\log n)$-competitive, as desired. This also completes the proof of Theorem 1.3.

5 Batch-Oblivious Delay Costs

We now turn to the batch-oblivious model, where the delay cost is defined globally over the vector of packet delays, without reference to how packets are grouped in acknowledgments. Formally, the objective is to minimize $k + f(d)$, where k is the number of acknowledgments, $f : \mathbb{R}_+^n \to \mathbb{R}_+$ is a monotone delay function, and $d = (d_1, \ldots, d_n)$ is the vector of waiting times, where d_i denotes how long packet i waits before being acknowledged. The function f is revealed incrementally as packets arrive. While this framework captures natural objectives such as total waiting time, it also includes more general functions, such as concave or submodular delay costs. As discussed earlier, greedy algorithms that succeed for additive delay fail in these settings, due to the ability of the optimal solution to concentrate delay on a few early packets and reduce marginal costs for the rest. A formal lower bound for concave delay functions is provided in Section 5.1. Here, we focus on broader classes of delay functions, including submodular functions and norms.

5.1 Concave Delay Cost

▶ **Theorem 1.6.** *Every deterministic algorithm is $\Omega(\sqrt{n})$-competitive for TCP Acknowledgment with batch-oblivious concave delay cost.*

Proof. Let $\ell \leq n/2$ and $\epsilon \in [0, 1]$ be parameters that will be set later. Define $x, y \in \mathbb{R}^n$ such that: each of the first ℓ coordinates of x is ϵ and each of the remaining coordinates is 1; each of the first ℓ coordinates of y is n/ℓ and each of the remaining is ϵ. Given a waiting time vector d, define $d_{\leq \ell} = \sum_{i \leq \ell} d_i$ and $d_{>\ell} = \sum_{i>\ell} d_i$. We now define the delay function

$$f(d) = \min\{d \cdot x, d \cdot y\} = \min\left\{\epsilon d_{\leq \ell} + d_{>\ell}, \frac{n}{\ell} d_{\leq \ell} + \epsilon d_{>\ell}\right\}.$$

Note that the function f is concave as it is the point-wise minimum of linear functions.

The adversary proceeds as follows. For the first ℓ packets, it releases packet i at time i. Let A_ℓ be the number of packets made by the algorithm just before time $\ell + 1$. If $A_\ell \geq \ell/2$ (case 1), it releases the remaining $n - \ell$ packets together at time $\ell + 1$; otherwise (case 2), it continues releasing each remaining packet i at time i.

We now show that in both cases, the adversary can construct a significantly cheaper solution. Consider case 1: the algorithm incurs a cost of at least $\ell/2$. Meanwhile, the adversary can use a single acknowledgment at time $\ell + 1$. Let d^* denote the resulting waiting time vector, where $d_i^* = \ell + 1 - i$ for $i \leq \ell$, and $d_i^* = 0$ for $i > \ell$. The delay cost of this solution is at most $f(d^*) \leq \epsilon \sum_{i=1}^{\ell} i = O(\ell^2 \epsilon)$, and the total cost is $1 + O(\ell^2 \epsilon)$. Therefore, the competitive ratio of the algorithm is $\Omega\big(\min\{\ell, 1/(\ell\epsilon)\}\big)$ in this case.

Next, we consider case 2. Let k denote the total number of acknowledgments made by the algorithm. Without loss of generality, we may assume that each acknowledgment occurs at the arrival time of some packet, as this can only reduce the total cost. Recall that in this case, packets arrive at unit times $1, 2, \ldots, n$. Thus, for any time interval starting at a packet arrival, the number of acknowledgments made during the interval plus the total waiting time incurred by the packets arriving within it must be at least the length of the interval. This is because the acknowledgment cost is 1, so each acknowledgment can "pay for" at most one unit of waiting time.

Now, consider the first ℓ packets. By assumption, the algorithm makes $A_\ell < \ell/2$ acknowledgments during this interval. From our previous observation, it follows that the total delay incurred by these packets is at least $\ell - A_\ell > \ell/2$. Thus, we obtain that $d \cdot y \geq (n/\ell) \cdot (\ell/2) = \Omega(n)$. On the other hand, consider only the remaining $n - \ell$ packets. Here, the number of acknowledgments and the overall waiting time sum up to $(k - A_\ell) + d_{>\ell}$. As noted above, this sum must be at least the length of the interval $(\ell, n]$, which is greater than $n/2$, since $\ell < n/2$. Hence, we have $k + d \cdot x \geq (k - A_\ell) + d_{>\ell} = \Omega(n)$. Thus, the total cost of the algorithm satisfies $k + \min\{d \cdot x, d \cdot y\} \geq \Omega(n)$.

On the other hand, consider the solution that acknowledges each of the first ℓ packets on arrival and makes one acknowledgment at the end at time n. Let d^* be its waiting time vector. Then, it holds that $f(d^*) \leq d^* \cdot y = \epsilon d_{>\ell} \leq O(n^2\epsilon)$ and so the solution has total cost $\ell + O(n^2\epsilon)$. Thus, the competitive ratio is $\Omega(\min\{n/\ell, 1/(n\epsilon)\})$ in this case.

Setting $\ell = \lceil \sqrt{n} \rceil$ and $\epsilon = 1/n^2$ gives the theorem. ◀

5.2 Submodular Delay Cost and Norms

We begin by defining continuous submodularity.

▶ **Definition 5.1** (Continuous submodular functions). *A function $f : \mathbb{R}^n_+ \to \mathbb{R}_+$ is* continuous submodular *if for every $x, y \in \mathbb{R}^n$, it satisfies $f(x \vee y) + f(x \wedge y) \leq f(x) + f(y)$, where $\vee$ denotes coordinate-wise maximum and $\wedge$ denotes coordinate-wise minimum.*

We remark that, unlike the case of discrete submodular functions, continuous submodularity is only equivalent to a weaker notion of diminishing marginals: for every $x, y \in \mathbb{R}^n_+$ such that $x \leq y$, $i \in [n]$ such that $x_i = y_i$ and $a \geq 0$, we have $f(x + ae_i) - f(x) \geq f(y + ae_i) - f(y)$, where e_i is the i-th standard basis vector. We refer the reader to [47, Remark 7 and Appendix A.1] for further discussion on this topic.

In fact, our analysis only uses an even weaker condition, where the diminishing-marginals property is required only when the coordinate being increased is zero in both vectors. We say that f satisfies *zero-coordinate diminishing marginals* if for every $x, y \in \mathbb{R}^n_+$ such that $x \leq y$, every $i \in [n]$ such that $x_i = y_i = 0$, and every $a \geq 0$, we have $f(x + ae_i) - f(x) \geq f(y + ae_i) - f(y)$. Clearly, continuous submodularity implies this condition.

For a vector $\Delta \in \mathbb{R}^n_+$, let $\mathrm{supp}(\Delta) = \{i \in [n] : \Delta_i > 0\}$. We will use the following simple observation.

▶ **Observation 5.2.** *Suppose that f satisfies zero-coordinate diminishing marginals. Let $u, v, \Delta \in \mathbb{R}^n_+$ be such that $u \leq v$ and $u_i = v_i = 0$ for every $i \in \mathrm{supp}(\Delta)$. Then*

$$f(u + \Delta) - f(u) \geq f(v + \Delta) - f(v).$$

Indeed, one can add the coordinates in $\mathrm{supp}(\Delta)$ one at a time. Before each coordinate is added, it is still zero in both the smaller and the larger vector, so zero-coordinate diminishing marginals apply. Summing these inequalities over all coordinates in $\mathrm{supp}(\Delta)$ gives the desired inequality.

▶ **Theorem 1.4.** *Greedy is a deterministic 2-competitive algorithm for TCP Acknowledgment with batch-oblivious continuous submodular delay cost.*

Proof. Recall that Greedy makes an acknowledgment whenever its delay cost increases by 1. We now analyze the competitive ratio of the algorithm. First, observe that by construction, the cost of the algorithm is at most twice the number of acknowledgments. Let k be the number of acknowledgments made by the algorithm. Fix an optimal solution. In the rest of the proof, we argue that its cost is at least k.

Let t_j be the time when the algorithm made its j-th acknowledgment, and define $T_j = (t_{j-1}, t_j]$ with $t_0 = 0$. For each packet i, let $d_i(j)$ and $d_i^*(j)$ be the amount of waiting time that packet i incurred up till t_j in the algorithm's solution and the optimal solution, respectively. If packet i arrived after t_j, we define $d_i(j)$ and $d_i^*(j)$ to be both 0.

At a high level, we adopt the proof framework of [29] for the standard TCP Acknowledgment setting and generalize it to our case: within each interval T_j, the optimal solution either issues an acknowledgment and pays a cost of 1 or its delay cost increases by at least 1. However, if the optimal solution allows earlier packets to wait a long time, it can effectively "front-load" its delay cost, which causes the marginal cost in later intervals to be too small. To address this, we introduce a capped version of the optimal delay vector, denoted $\bar{d}$, where each entry is defined as $\bar{d}_i(j) = \min\{d_i^*(j), d_i(j)\}$. In what follows, we use $d(j)$ and $\bar{d}(j)$ to denote the vectors $(d_1(j), \ldots, d_n(j))$ and $(\bar{d}_1(j), \ldots, \bar{d}_n(j))$, respectively. By monotonicity, the optimal solution's delay cost $f(d^*) \geq f(\bar{d})$ and so it suffices to show that if the optimal solution makes no acknowledgment in T_j, then the increase in the delay cost of the capped optimal delay vector in T_j is $f(\bar{d}(j)) - f(\bar{d}(j-1)) \geq 1$.

▷ Claim 5.3. For each interval T_j, if the optimal solution did not make an acknowledgment in T_j, then

$$f(\bar{d}(j)) - f(\bar{d}(j-1)) \geq f(d(j)) - f(d(j-1)) = 1.$$

Proof. The equality follows from the fact that $f(d(j)) - f(d(j-1))$ is the increase in the algorithm's delay cost during interval T_j, and the algorithm makes an acknowledgment exactly when its delay cost increases by 1 since the last acknowledgment.

It remains to prove the inequality. Let $\Delta^j = d(j) - d(j-1)$ be the increase in the algorithm's delay vector during interval T_j. Since the algorithm acknowledged all pending packets at time t_{j-1}, the only packets whose delay can increase during T_j are the packets that arrive during T_j. Hence, for every $i \in \mathrm{supp}(\Delta^j)$, we have $d_i(j-1) = 0$. Moreover, by the definition of the capped vector, $\bar{d}_i(j-1) = 0$. We also have $\bar{d}(j-1) \leq d(j-1)$. Therefore, applying Observation 5.2 with $u = \bar{d}(j-1)$, $v = d(j-1)$, and $\Delta = \Delta^j$, we get

$$f(\bar{d}(j-1) + \Delta^j) - f(\bar{d}(j-1)) \geq f(d(j-1) + \Delta^j) - f(d(j-1)).$$

Since $d(j-1) + \Delta^j = d(j)$, this gives

$$f(\bar{d}(j-1) + \Delta^j) - f(\bar{d}(j-1)) \geq f(d(j)) - f(d(j-1)). \tag{2}$$

It remains to compare $\bar{d}(j-1) + \Delta^j$ with $\bar{d}(j)$. We claim that $\bar{d}(j-1) + \Delta^j \leq \bar{d}(j)$ coordinatewise. Fix a packet i. If i arrived during T_j, then, since the optimal solution did not make an acknowledgment in T_j, the packet has the same waiting time up till t_j in both solutions. Thus, $d_i^*(j) = d_i(j)$, and hence $\bar{d}_i(j) = d_i(j) = \Delta_i^j$. Also, $\bar{d}_i(j-1) = 0$, so the desired inequality holds with equality.

If packet i arrived no later than t_{j-1}, then the algorithm does not incur additional waiting time for packet i during T_j, so $\Delta_i^j = 0$. Since waiting times in the optimal solution do not decrease while no acknowledgment is made, and since $d_i(j) = d_i(j-1)$, we have

$\bar{d}_i(j-1) \le \bar{d}_i(j)$. Thus, the desired inequality again holds. Finally, if packet i arrives after t_j, all relevant quantities are 0, and the inequality is trivial. Therefore, $\bar{d}(j-1) + \Delta^j \le \bar{d}(j)$, as desired.

Applying monotonicity of f to the above inequality, we have $f(\bar{d}(j)) \ge f(\bar{d}(j-1) + \Delta^j)$. Plugging this into Equation (2) gives

$$f(\bar{d}(j)) - f(\bar{d}(j-1)) \ge f(d(j)) - f(d(j-1)).$$

Together with the greedy rule $f(d(j)) - f(d(j-1)) = 1$, this proves the claim. ◁

Suppose the optimal solution makes k^* acknowledgments. If $k^* \ge k$, then we are done. Otherwise, there exist at least $k - k^*$ intervals in which the optimal solution did not make an acknowledgment. By monotonicity of f and Claim 5.3, the optimal solution's delay cost is

$$f(d^*) \ge f(\bar{d}) = \sum_{j=1}^{k} f(\bar{d}(j)) - f(\bar{d}(j-1)) \ge k - k^*.$$

Thus, the optimal solution's total cost is at least k. ◀

Patton, Russo, and Singla [47] observed that ℓ_p norms, Top-k norms and ordered norms are submodular. They also proved a result on the approximability of symmetric norms by submodular norms. Using these results give us the following corollary.

▶ **Corollary 1.5.** *There is a deterministic 2-competitive algorithm for TCP Acknowledgment, where the delay cost is: an ℓ_p norm, a Top-ℓ norm, or an ordered norm. When the delay cost is a symmetric norm $\|\cdot\|$, then there is a deterministic $O(\log \rho)$-competitive algorithm where $\rho = \|(1,1,\ldots,1)\| \,/\, \|(1,0,\ldots,0)\| \le n$. In particular, there is a deterministic $O(\log n)$-competitive algorithm when the delay cost is a monotone symmetric norm.*

▶ Remark 5.4. The zero-coordinate diminishing-marginals condition has a natural modeling interpretation in practical domains beyond TCP Acknowledgment. It says that the marginal cost of making a new coordinate positive is smaller when more other coordinates are already present. This is appropriate for incident-based or SLA-type[8] delay costs, where the first few delayed requests may create most of the operational or reputational cost, while additional newly delayed requests are partly absorbed into the same incident. For example, functions of the form $f(d) = g(\mathrm{supp}(d))$, where g is a monotone submodular function, satisfy zero-coordinate diminishing marginals: adding a new positive coordinate corresponds exactly to adding a new element to a larger set, whose marginal contribution is smaller by the submodularity of g. Such functions model global costs depending on the set of requests that experience any positive delay, rather than on the precise magnitude of each delay.

6 Conclusion

In this work, we revisit Online TCP Acknowledgment under broad and natural generalizations of delay costs. We show that the classical guarantees of Greedy extend well beyond additive delays: it remains 2-competitive for *max-monotone* objectives and for a rich class of *batch-oblivious* continuous submodular delay costs. At the same time, we identify a sharp boundary where this paradigm breaks, by proving that Greedy can be $\Omega(n)$-competitive for *sum-monotone* delays. For this harder regime, we develop a new phase-based online algorithm

[8] SLA = Service Level Agreement.

that achieves a tight $\Theta(\log n)$ competitive ratio. Together, our results give a more complete picture of TCP under non-additive delay objectives and lay conceptual groundwork for online problems with delay more broadly.

There are several intriguing open problems. First, what is the competitiveness of other problems such as Joint Replenishment, Multi-Level Aggregation, and Set Cover with Delay under the more general delay cost models considered in this work? Second, while we have a tight characterization of the deterministic competitive ratio for sum-monotone delays, it is unclear whether randomization can help. For the Parking Permit problem, Meyerson [44] showed that the randomized competitive ratio is $\Theta(\log \log n)$. A reduction from Parking Permit will imply the same lower bound. Finally, are there other interesting delay objectives? For example, fairness-oriented delay objectives, where the goal is to balance efficiency with equitable treatment of packets, a setting that remains largely unexplored in the TCP acknowledgment framework.

References

1 Susanne Albers. Online algorithms: a survey. *Mathematical Programming*, 97:3–26, 2003. doi:10.1007/S10107-003-0436-0.

2 Susanne Albers and Helge Bals. Dynamic TCP acknowledgement: penalizing long delays. In *Proceedings of the Fourteenth Annual ACM-SIAM Symposium on Discrete Algorithms, January 12-14, 2003, Baltimore, Maryland, USA*, pages 47–55. ACM/SIAM, 2003. URL: http://dl.acm.org/citation.cfm?id=644108.644116.

3 Susanne Albers and Helge Bals. Dynamic TCP acknowledgment: Penalizing long delays. *SIAM J. Discret. Math.*, 19(4):938–951, 2005. doi:10.1137/S0895480104441656.

4 Mark Allman. On the generation and use of tcp acknowledgments. *ACM SIGCOMM Computer Communication Review*, 28(5):4–21, 1998. doi:10.1145/303297.303301.

5 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. 81:1:1–1:20, 2017. doi:10.4230/LIPIcs.APPROX-RANDOM.2017.1.

6 Yossi Azar, Ashish Chiplunkar, and Haim Kaplan. Polylogarithmic bounds on the competitiveness of min-cost perfect matching with delays. In Philip N. Klein, editor, *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2017, Barcelona, Spain, Hotel Porta Fira, January 16-19*, pages 1051–1061. SIAM, 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 *28th Annual European Symposium on Algorithms, ESA 2020, September 7-9, 2020, Pisa, Italy (Virtual Conference)*, volume 173 of *LIPIcs*, pages 8:1–8:21. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2020. doi:10.4230/LIPIcs.ESA.2020.8.

8 Yossi Azar, Arun Ganesh, Rong Ge, and Debmalya Panigrahi. Online service with delay. In *Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing*, pages 551–563, 2017. doi:10.1145/3055399.3055475.

9 Yossi Azar and Shahar Lewkowicz. Online joint replenishment problem with arbitrary holding and backlog costs. In *Proceedings of the 2026 ACM-SIAM Symposium on Discrete Algorithms, SODA 2026, Vancouver, Canada, January 11-14, 2026*, pages 2702–2725. SIAM, 2026. doi:10.1137/1.9781611978971.99.

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

11 Yossi Azar and Noam Touitou. General framework for metric optimization problems with delay or with deadlines. In *Proceedings of the 60th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2019*, pages 60–71. IEEE Computer Society, 2019. doi:10.1109/FOCS.2019.00013.

12 Yossi Azar and Noam Touitou. Beyond tree embeddings–a deterministic framework for network design with deadlines or delay. In *2020 IEEE 61st Annual Symposium on Foundations of Computer Science (FOCS)*, pages 1368–1379. IEEE, 2020. doi:10.1109/FOCS46700.2020.00129.

13 Francis Bach. Submodular functions: from discrete to continuous domains. *Mathematical Programming*, 175(1):419–459, 2019. doi:10.1007/s10107-018-1248-6.

14 Marcin Bienkowski, Martin Böhm, Jaroslaw Byrka, Marek Chrobak, Christoph Dürr, Lukáš Folwarcznỳ, Łukasz Jeż, Jiří Sgall, Nguyen Kim Thang, and Pavel Veselỳ. Online algorithms for multilevel aggregation. *Operations Research*, 68(1):214–232, 2020. doi:10.1287/OPRE.2019.1847.

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

16 Marcin Bienkowski, Jaroslaw Byrka, Marek Chrobak, Lukasz Jez, Dorian Nogneng, and Jirí Sgall. Better approximation bounds for the joint replenishment problem. In *Proceedings of the Twenty-Fifth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2014, Portland, Oregon, USA, January 5-7, 2014*, pages 42–54. SIAM, 2014. doi:10.1137/1.9781611973402.4.

17 Marcin Bienkowski, Artur Kraska, and Paweł Schmidt. A match in time saves nine: Deterministic online matching with delays. In *International Workshop on Approximation and Online Algorithms*, pages 132–146. Springer, 2017.

18 Marcin Bienkowski, Artur Kraska, and Pawel Schmidt. Online service with delay on a line. In Zvi Lotker and Boaz Patt-Shamir, editors, *Structural Information and Communication Complexity - 25th International Colloquium, SIROCCO 2018, Ma'ale HaHamisha, Israel, June 18-21, 2018, Revised Selected Papers*, Lecture Notes in Computer Science, pages 237–248. Springer, 2018. doi:10.1007/978-3-030-01325-7_22.

19 Carlos Brito, Elias Koutsoupias, and Shailesh Vaya. Competitive analysis of organization networks or multicast acknowledgement: how much to wait? In *Proceedings of the Fifteenth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2004, New Orleans, Louisiana, USA, January 11-14, 2004*, pages 627–635. SIAM, 2004. URL: http://dl.acm.org/citation.cfm?id=982792.982886.

20 Niv Buchbinder, Moran Feldman, Joseph (Seffi) Naor, and Ohad Talmon. O(depth)-competitive algorithm for online multi-level aggregation. In *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2017, Barcelona, Spain, Hotel Porta Fira, January 16-19*, pages 1235–1244. SIAM, 2017. doi:10.1137/1.9781611974782.80.

21 Niv Buchbinder, Tracy Kimbrel, Retsef Levi, Konstantin Makarychev, and Maxim Sviridenko. Online make-to-order joint replenishment model: Primal-dual competitive algorithms. *Oper. Res.*, 61(4):1014–1029, 2013. doi:10.1287/OPRE.2013.1188.

22 Rodrigo A. Carrasco, Kirk Pruhs, Cliff Stein, and José Verschae. The online set aggregation problem. In *LATIN 2018: Theoretical Informatics - 13th Latin American Symposium, Buenos Aires, Argentina, April 16-19, 2018, Proceedings*, volume 10807 of *Lecture Notes in Computer Science*, pages 245–259. Springer, 2018. doi:10.1007/978-3-319-77404-6_19.

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

24 David D Clark. Rfc0813: Window and acknowledgement strategy in tcp, 1982.

25 Ruy De Oliveira and Torsten Braun. A dynamic adaptive acknowledgment strategy for tcp over multihop wireless networks. In *Proceedings IEEE 24th Annual Joint Conference of the IEEE Computer and Communications Societies.*, volume 3, pages 1863–1874. IEEE, 2005. doi:10.1109/INFCOM.2005.1498465.

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

27 Michael Dinitz, Jeremy T. Fineman, and Seeun William Umboh. Learning-augmented online algorithms for nonclairvoyant joint replenishment problem with deadlines. In Sayan Bhattacharya, Danupon Nanongkai, Michael Benedikt, and Gabriele Puppis, editors, *53rd International Colloquium on Automata, Languages, and Programming, ICALP 2026, Royal Holloway, University of London, Egham, United Kingdom, July 7-10, 2026*, volume 374 of *LIPIcs*, pages 77:1–77:21. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2026. doi:10.4230/LIPIcs.ICALP.2026.77.

28 Daniel R Dooly, Sally A Goldman, and Stephen D Scott. Tcp dynamic acknowledgment delay (extended abstract) theory and practice. In *Proceedings of the thirtieth annual ACM symposium on Theory of computing*, pages 389–398, 1998.

29 Daniel R Dooly, Sally A Goldman, and Stephen D Scott. On-line analysis of the tcp acknowledgment delay problem. *Journal of the ACM (JACM)*, 48(2):243–273, 2001. doi:10.1145/375827.375843.

30 Marc Dufay and Roger Wattenhofer. A deterministic polylogarithmic competitive algorithm for matching with delays. In Kasper Green Larsen and Barna Saha, editors, *Proceedings of the 2026 Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2026, Vancouver, BC, Canada, January 11-14, 2026*, pages 3936–3964. SIAM, 2026. doi:10.1137/1.9781611978971.144.

31 Yuval Emek, Shay Kutten, and Roger Wattenhofer. Online matching: haste makes waste! In *Proceedings of the forty-eighth annual ACM symposium on Theory of Computing*, pages 333–344, 2016. doi:10.1145/2897518.2897557.

32 Yuval Emek, Yaacov Shapiro, and Yuyi Wang. Minimum cost perfect matching with delays for two sources. *Theor. Comput. Sci.*, 754:122–129, 2019. doi:10.1016/J.TCS.2018.07.004.

33 Tomer Ezra, Stefano Leonardi, Michal Pawlowski, Matteo Russo, and Seeun William Umboh. Universal optimization for non-clairvoyant subadditive joint replenishment. *Algorithmica*, 88(3):43, 2026. doi:10.1007/S00453-026-01389-1.

34 Junhao Gan, Xiao Sun, and Seeun William Umboh. Online matching with size-based and convex delays. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques, APPROX/RANDOM 2026*, 2026. To appear.

35 Anupam Gupta, Amit Kumar, and Debmalya Panigrahi. A hitting set relaxation for k-server and an extension to time-windows. In *62nd IEEE Annual Symposium on Foundations of Computer Science, FOCS 2021, Denver, CO, USA, February 7-10, 2022*, pages 504–515. IEEE, 2021. doi:10.1109/FOCS52979.2021.00057.

36 Anupam Gupta, Amit Kumar, and Debmalya Panigrahi. Caching with time windows and delays. *SIAM J. Comput.*, 51(4):975–1017, 2022. doi:10.1137/20M1346286.

37 Péter Györgyi, Tamás Kis, Tímea Tamási, and József Békési. Joint replenishment meets scheduling. *Journal of Scheduling*, 26(1):77–94, 2023. doi:10.1007/S10951-022-00768-0.

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

39 Anna R. Karlin, Claire Kenyon, and Dana Randall. Dynamic TCP Acknowledgment and Other Stories about e/(e - 1). *Algorithmica*, 36(3):209–224, July 2003. doi:10.1007/s00453-003-1013-x.

40 Anna R. Karlin, Mark S. Manasse, Larry Rudolph, and Daniel Dominic Sleator. Competitive snoopy caching. *Algorithmica*, 3:77–119, 1988. doi:10.1007/BF01762111.

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

42 Ngoc Mai Le, Seeun William Umboh, and Ningyuan Xie. The power of clairvoyance for multi-level aggregation and set cover with delay. In *Proceedings of the 2023 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1594–1610. SIAM, 2023. doi:10.1137/1.9781611977554.CH59.

43 Mathieu Mari, Michal Pawlowski, Runtian Ren, and Piotr Sankowski. Online matching with delays and stochastic arrival times. *Theory Comput. Syst.*, 69(1):12, 2025. doi:10.1007/S00224-024-10207-6.

44 Adam Meyerson. The parking permit problem. In *46th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2005, Pittsburgh, PA, USA, October 23-25, 2005, Proceedings*, pages 274–284. IEEE Computer Society, 2005. doi:10.1109/SFCS.2005.72.

45 Benjamin Moseley, Aidin Niaparast, and R Ravi. Putting off the catching up: Online joint replenishment problem with holding and backlog costs. In *Proceedings of the 2025 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 3865–3883. SIAM, 2025. doi:10.1137/1.9781611978322.130.

46 Joseph (Seffi) Naor, Seeun William Umboh, and David P. Williamson. Tight bounds for online weighted tree augmentation. *Algorithmica*, 84(2):304–324, 2022. doi:10.1007/S00453-021-00888-7.

47 Kalen Patton, Matteo Russo, and Sahil Singla. Submodular norms with applications to online facility location and stochastic probing. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques, APPROX/RANDOM 2023, September 11-13, 2023, Atlanta, Georgia, USA*, volume 275 of *LIPIcs*, pages 23:1–23:22. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2023. doi:10.4230/LIPIcs.APPROX/RANDOM.2023.23.

48 Steven S. Seiden. A guessing game and randomized online algorithms. In *Proceedings of the Thirty-Second Annual ACM Symposium on Theory of Computing, May 21-23, 2000, Portland, OR, USA*, pages 592–601. ACM, 2000. doi:10.1145/335305.335385.

49 David Shmoys, Varun Suriyanarayana, and Seeun William Umboh. Improved online algorithms for inventory management problems with holding and delay costs: Riding the wave makes things simpler, stronger, & more general. In *Proceedings of the 2026 ACM-SIAM Symposium on Discrete Algorithms, SODA 2026, Vancouver, Canada, January 11-14, 2026*, pages 2726–2742. SIAM, 2026. doi:10.1137/1.9781611978971.100.

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

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

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

53 Noam Touitou. Nearly-optimal algorithm for non-clairvoyant service with delay. In Olaf Beyersdorff, Michal Pilipczuk, Elaine Pimentel, and Kim Thang Nguyen, editors, *42nd International Symposium on Theoretical Aspects of Computer Science, STACS 2025, Jena, Germany, March 4-7, 2025*, LIPIcs, pages 74:1–74:21. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2025. doi:10.4230/LIPIcs.STACS.2025.74.