Online Dynamic Acknowledgement with Learned Predictions

Sungjin Im
Electrical Engineering and Computer Science
University of California at Merced
Merced, US
sim3@ucmerced.edu

Benjamin Moseley
Tepper School of Business
Carnegie Mellon University
Pittsburgh, US
moseleyb@andrew.cmu.edu

Chenyang Xu
Software Engineering Institute
East China Normal University
Shanghai, China
xcy1995@zju.edu.cn

Ruilong Zhang
Department of Computer Science
City University of Hong Kong
Hong Kong, China
ruilzhang4-c@my.cityu.edu.hk

IEEE INFOCOM 2023 - IEEE Conference on Computer Communications | 979-8-3503-3414-2/23/$31.00 ©2023 IEEE | DOI: 10.1109/INFOCOM53939.2023.10228882

***Abstract*—We revisit the online dynamic acknowledgment problem. In the problem, a sequence of requests arrive over time to be acknowledged, and all outstanding requests can be satisfied simultaneously by one acknowledgement. The goal of the problem is to minimize the total request delay plus acknowledgement cost. This elegant model studies the trade-off between acknowledgement cost and waiting experienced by requests. The problem has been well studied and the tight competitive ratios have been determined. For this well-studied problem, we focus on how to effectively use machine-learned predictions to have better performance.**

We develop algorithms that perform arbitrarily close to the optimum with accurate predictions while concurrently having the guarantees arbitrarily close to what the best online algorithms can offer without access to predictions, thereby achieving simultaneous optimum consistency and robustness. This new result is enabled by our novel prediction error measure. No error measure was defined for the problem prior to our work, and natural measures failed due to the challenge that requests with different arrival times have different effects on the objective. We hope our ideas can be used for other online problems with temporal aspects that have been resisting proper error measures.

***Index Terms*—Online Algorithms, Competitive Ratio, Approximation Algorithms, Learning-augmented Algorithms**

I. Introduction

In a typical communication setting where a client receives a sequence of packets from the server, she needs to acknowledge the receipt of the packets to update the server regarding the current communication status. There are two desirable goals in conflict. On the one hand, the server would like to get prompt feedback from the client, which means the client should make more acknowledgements. On the other hand, acknowledging frequently incurs a huge communication cost, and therefore it is desirable to make fewer acknowledgements, which results in prolonged latency in feedback. Thus, there is a fundamental trade-off between making fewer acknowledgements and reducing acknowledgement latency.

The online dynamic acknowledgement problem[1] (DAP) is an elegant model that was introduced in [17] to study the above trade-off of fundamental importance. An instance of DAP is a sequence of n requests (or demands) arriving online. Succinctly, it can be represented as $(p_t)_{t\in[T]}$, where p_t is the number of demands (equivalently packets or requests) that arrive at time $t \in [T]$ and it is unknown to the algorithm. The requests must be acknowledged. When the client acknowledges (acks for short), all outstanding[2] requests are simultaneously satisfied. An outstanding request incurs $1/d$ delay cost each time, where d is an input parameter, and each ack costs 1. The objective is to minimize the total ack cost plus the delay cost of all requests.

The DAP admits a simple 2-competitive[3] greedy algorithm [17] that acks when the outstanding requests have accumulated delay cost equal to 1. Also there is a $\frac{e}{e-1}$-competitive randomized algorithm [13], [20], [32]. The competitive ratios are tight for both deterministic and randomized algorithms. The offline version of the problem can be solved optimally via dynamic programming or linear programming as the LP has no integrality gap [13].

While DAP is well understood, the traditional study of online algorithms using competitive ratios is often criticized for its pessimistic view of the instances. On the one hand, optimizing the competitive ratio gives robust guarantees against any possible inputs. On the other hand, algorithms that optimize

Chenyang Xu was supported in part by Science and Technology Innovation 2030 –"The Next Generation of Artificial Intelligence" Major Project No.2018AAA0100900. Sungjin Im was supported in part by NSF grants CCF-1844939 and CCF-2121745. Benjamin Moseley was supported in part by a Google Research Award, an Infor Research Award, a Carnegie Bosch Junior Faculty Chair and NSF grants CCF-2121744 and CCF-1845146

All authors (ordered alphabetically) have equal contributions. Correspondence to Chenyang Xu, Ruilong Zhang, Benjamin Moseley and Sungjin Im.

[1]This problem is also called the Dynamic TCP Acknowledgement problem.

[2]A request is said to be outstanding if it has arrived yet has not been acknowledged (or equivalently satisfied).

[3]An online algorithm is said to be c-competitive if its objective is at most c times the optimum for all inputs.

the competitive ratio could be highly tailored toward working well against worst-case instances, sacrificing performance for typical instances that tend not to be adversarial.

The framework of augmenting discrete optimization algorithms with machine learning [21], [26] has recently emerged as a powerful framework for algorithm analysis. Such algorithms leverage machine learned parameters to give beyond-worst case performance guarantees while providing robustness guarantees even when given inaccurate parameters from the machine learning. The goal is to develop algorithms that perform extremely well using ML for typical instances and exhibit robustness against exceptional—even adversarial—instances as traditional worst-case algorithms do.

This model has been used for various online problems. The predicted parameters can be used to cope with the uncertainty in the input. For example, caching [19], [26], [31], [34], buy-or-rent [2], [22], load balancing [23], [24], scheduling [8], [18], [22], [28], secretary problem [4], metrical task systems [3], to name a few.

ML augmented algorithms typically take an input I and a prediction P on the input. The prediction may be revealed at the beginning or gradually over time as the input is. There is an error function $\eta(I, P)$ defined to measure the quality of the prediction. The prediction is of high quality when η is small. The algorithm's objective—if it is to be minimized—is commonly bounded by a quantity of the following form:

$$\min\{\alpha \mathrm{OPT}(I) + \beta\eta(I, P), \gamma \mathrm{OPT}(I)\}, \tag{1}$$

where $\mathrm{OPT}(I)$ denotes the optimal objective on input I. The algorithm is then said to be α-*consistent* and γ-*robust*. In other words, the algorithm is almost α-competitive when the prediction is very accurate and always at most γ-competitive simultaneously.

This paper seeks to study the DAP assuming we have learned predictions on the arriving requests. Specifically, taking access to prediction, $(\hat{p}_t)_{t\in[T]}$, we would like to achieve consistency and robustness for a certain error measure η.

a) Simultaneous Optimum Consistency and Robustness: The above parameters, α, β, and γ are correlated, and the guarantees differ depending on the correlation. Ideally, the guarantees should have the following form.

$$\min\{(1+\epsilon)\mathrm{OPT} + c\eta, (c^* + \epsilon)\mathrm{OPT}\}, \tag{2}$$

where c^* is the best competitive ratio that can be achieved without using predictions and c is a constant depending on ϵ. That is, we would like to achieve two goals simultaneously: on the one hand, we achieve near optimality when the predictions are almost perfect; on the other hand, we simultaneously achieve the best robustness against any inputs regardless of the prediction quality. We say this guarantee is *simultaneous optimum consistency and robustness*.

Intuitively, this kind of guarantee can be achieved as follows. We have two algorithms, A that closely follows predictions and B that is robust against all inputs. If we know that η is large, then we use B—otherwise, A. However, η is a function that depends on the whole input I and the prediction P. Therefore, we can only estimate its value before seeing the entire input. Of course, one can design a trivial algorithm that uses B as soon as she notices that the prediction is not perfect. However, the algorithm will then rarely benefit from predictions. Thus, it is critical to define an error measure that grows graciously as the actual instance deviates from the predictions to be able to develop an algorithm that still outperforms B for mild prediction errors. The algorithm crucially relies on the error measure η.

The primary goal of this paper is to develop an ML-augmented algorithm for the DAP, which seems to resist a reasonable definition of η. Further, it is one of the most fundamental online problems with temporal aspects where defining a good error measure has been elusive.

A. Critical Need for Prediction Error

Bamas *et al.* [9] gave a very elegant framework to smoothly combine various primal-dual algorithms with an arbitrary solution. In particular, for DAP, they assume access to a complete solution as advice and let the primal-dual algorithm mimic the prediction. At a high level, they increase each outstanding request's 'potential' and acks when their aggregated potential justifies it. If the algorithm is behind the prediction for a request, it increases its potential more aggressively to catch up with it. The combined algorithm has cost at most $\min\{\frac{\lambda}{1-e^{-\lambda}}A, \frac{1}{1-e^{-\lambda}}\mathrm{OPT}\}$ for any $\lambda \in (0, 1]$, where A is cost of the given solution on the instance.

Unfortunately, the result has two critical issues. First, it assumes that we are given a complete solution for the input as advice. Thus, if we are only given the number of demands at each time as a prediction, we still have to devise an algorithm. It is possible to use the optimum solution for the predicted instance as a solution for the actual instance, although it is unclear if it is the best way to use the prediction. Another issue is that it does not provide simultaneous optimum consistency and robustness: To achieve near-optimum consistency, one cannot help but make $\frac{\lambda}{1-e^{-\lambda}} \to 1$, but it will make $\frac{1}{1-e^{-\lambda}} \to \infty$, resulting a poor robust guarantee. The error measure serves as a barometer for the prediction's accuracy, and the algorithm cannot change its behavior agilely without it.

B. Challenges in Defining Prediction Error

Despite the critical need for an error measure, it is non-trivial to define for DAP—more broadly, problems that involve temporal aspects; see Section I-D. This is because requests with different arrival times could have different contributions to the objective, and the interaction between the delay cost and the ack cost is subtle. For instance, a few requests arriving later may make the optimal solution switch most of its cost from acks to delay cost and change the solution structure.

Intuitively, if the error is small, there should be a solution that is simultaneously good for both the actual and predicted instances. It is not difficult to see that naive error measures fail to satisfy it. For example, say we use the ℓ_1-norm, the aggregate sum of the prediction error at each time, i.e.,

$\ell_1(I, \hat{I}) = \sum_{t\in[T]} |p_t - \hat{p}_t|$. But even if the instance is different from the prediction by only one request, the optimum can change a lot depending on whether the request is close to other requests. In short, this is because the ℓ_1-norm is oblivious to the arrival times. We discuss other natural error measures in Section II-B3 in detail and why they are unsatisfactory.

C. Our Contributions

Our contributions are summarized as follows.

1) We give *simultaneous optimum consistency and robustness* for DAP for the first time (Section III and Section IV).
2) We propose a *novel error measure*, which enables our algorithm and guarantees (Section II-B1).
3) We show that the predictions are *learnable* with respect to the error measure (Section V).
4) Our experiments show that our new algorithm beats known learning augmented algorithms and remains on par with the algorithms with the best competitive ratio at the most time (Section VI). The experiments demonstrate the theory is predictive of practice.

Our main theoretical result is the following for our new error η, which will be shortly described.

Theorem 1. *For any $\epsilon > 0$, there is a randomized algorithm whose objective is bounded by $\min\{(1+\epsilon)\mathrm{OPT} + O(\frac{1}{\epsilon^2})\eta, (\frac{e}{e-1}+\epsilon)\mathrm{OPT}\}$ in expectation. Further, there is a deterministic algorithm whose objective is bounded by $\min\{(1+\epsilon)\mathrm{OPT} + O(\frac{1}{\epsilon^2})\eta, (2+\epsilon)\mathrm{OPT}\}$.*

In other words, we obtain simultaneous optimum consistency and robustness both deterministically and randomly. We also complement this result by showing that no deterministic algorithms have a cost smaller than $\min\{(1+\lambda-\epsilon)\cdot \mathrm{OPT}(I), (1+\frac{\eta}{\lambda})\}$ for any constant $\lambda > 0$ and sufficiently small $\epsilon > 0$; see Theorem 4.

Our new guarantees and algorithm crucially rely on our novel error measure. At a high level, we use the optimal objective to define η, inspired by [18]. Following their idea, we can try to measure the difference between $\mathrm{OPT}((\max\{p_t, \hat{p}_t\})_t)$ and $\mathrm{OPT}((\min\{p_t, \hat{p}_t\})_t)$. The former (the latter, resp.) is the optimal objective assuming the number of requests is the maximum (minimum, resp.) of the actual number and the prediction at each time. Although this satisfies the desiderata proposed by them, Monotonicity and Lipschitzness,[4] it fails to capture the temporal aspects of the problem as requests with different arrival times could have a different effect; see Section II-B3 for more detail. In particular, we show it could mistakenly label some bad predictions as good, making the error unusable in guiding the algorithm's decision. Therefore, we partition the time horizon and aggregate the error measured in each sub-interval. The maximum aggregate error over all partitions is what we adopt.

[4]Monotonicity means the error should get smaller if the predictions are more correct; and Lipschitzness means the error should change as much as the objective to successfully distinguish between good and poor predictions.

Under the new error measure, we successfully design a novel algorithm that quickly switches between exploiting the predictions and running robust algorithms. The algorithm is subtle. At a high level, the algorithm first computes a nearly optimal solution that is stable in that adding extra acks cannot significantly reduce the cost. Intuitively this gives us an interval where we can measure the error without worrying too much about the interaction between the delay and ack costs. We set a budget we can use until the first time t_1 when the nearly optimal solution acks. Until the time t_1, at each time we ack if we are still within the budget and the optimal solution on the actual instance would ack right now. Note that we only loosely follow the prediction as the actual instance could be quite different from the prediction. Rather, we use the budget to figure out how much we should tolerate the errors. If we run out of the budget before t_1 we switch to a robust algorithm, which can be the 2-competitive greedy algorithm or the $\frac{e}{e-1}$-competitive randomized algorithm. The algorithm is recursively defined from t_1 or from the time it exhausts the budget.

The prediction $(\hat{p}_t)_{t\in[T]}$ we use in our paper is natural and provably learnable. We show that the best prediction can be learned from polynomially many samples in T if instances follow a certain unknown distribution. Due to space limits, some proofs are omitted and can be found in the full version.

D. Other Related Work

Balancing the communication cost and delay cost has been studied extensively due to its fundamental importance in communication network, such as multicast acknowledgment [10], [11], [15], [29]; broadcast scheduling [25], [33], etc. DAP is one of the elegant models that captures the trade-offs between communication cost and delay cost, and therefore it also has been studied in many previous works [1], [5], [14], [16].

Due to the explosive volume of work in the area of the learning-augmented algorithm, we only discuss the most closely related work. As discussed, [9] gave a framework that combines an arbitrary solution and a primal-dual algorithm for various problems, such as online set cover and DAP. More generally, [27] showed how to be competitive against two online algorithms simultaneously. In general, such approaches cannot achieve simultaneous optimum consistency and robustness.

We briefly discuss online problems with temporal aspects. To our knowledge, no work prior to ours assumes predictions on jobs or requests' arrival time. Our error measure is inspired in part by the recent work by [18] for non-clairvoyant scheduling where the goal is to better minimize average completion time using predictions on job processing times. However, their work assumes all jobs arrive at *the same time*. Its preceding work [30] uses the same prediction model but a different error measure. For average response time, see [6]. For various problems involving latency, see [7] and the pointers therein. For connection to inventory management problems, see [12].

II. Preliminaries & Prediction Error

A. Notations

To formally study the DAP, we set up some notations that will be used throughout the paper. We use a set of points in time to denote a solution $X = \{ x_1, \ldots, x_k \}$ where the time points in the set are sorted in increasing order. We say it is feasible to instance $I = (p_t)_{t \in T}$ if (i) $X \subseteq [T]$ and (ii) $\max X \geq \arg\max_t [p_t > 0]$. Let $n = \arg\max_t [p_t > 0]$. Let $x_0 = 0$ and let $\mathrm{F}(I, X)$ be the objective value of the solution X applied to instance I:

$$\mathrm{F}(I, X) = |X| + \frac{1}{d} \cdot \sum_{i=1}^{|X|} \Big(\sum_{t=x_{i-1}+1}^{x_i} p_t (x_i - t) \Big),$$

where each additional time unit of latency incurs a cost of $\frac{1}{d}$.

For an arbitrary instance I, let $\mathrm{OPT}(I)$ denote the optimal solution or its objective depending on the context. Similarly, let $\mathrm{ALG}(I)$ denote the solution of our algorithm, which will be discussed later, or its objective. Let $\mathrm{D}(I, X)$ and $\#(I, X)$ be the total delay cost and ack cost of the $\mathrm{F}(I, X)$.

In the analysis, we will frequently partition the time horizon $[1, T]$ according to a solution $X = \{ x_1, \ldots, x_k \}$. The partition $\mathcal{P}_X = (P_1, \ldots, P_k)$ is called a (time) partition induced by X if and only if $P_i = \{ t \mid x_{i-1} < t \leq x_i \}$ for all $i \in [k]$. Note that $\mathcal{P}$ is a partition of the time interval $[1, n]$.

B. Error Measure

Given an actual instance $I = (p_t)_{t \in [T]}$ of DAP and its predicted instance $\hat{I} = (\hat{p}_t)_{t \in [\hat{T}]}$, we would like to define a sound and effective error measure. Let $n = \max\{T, \hat{T}\}$. We can always assume that both the actual instance and the predicted instance have n time points by adding zero package points. As discussed in Section I-C, we would like to satisfy the Monotonicity and Lipschitzness properties proposed by [18]:

Definition 1 ([18]). *The error function* ERR *is monotone if for any* $S \subseteq [n]$,

$$\mathsf{ERR}\Big((p_t)_{t \in [n]}, (\hat{p}_t)_{t \in [n]} \Big) \geq \mathsf{ERR}\Big((p_t)_{t \in n}, (p_t)_{t \in [S]} \cup (\hat{p}_t)_{t \in [n] \setminus S} \Big),$$

while it has Lipschitzness if

$$|\mathrm{OPT}((p_t)_{t \in [n]}) - \mathrm{OPT}((\hat{p}_t)_{t \in [n]})| \leq \mathsf{ERR}\Big((p_t)_{t \in [n]}, (\hat{p}_t)_{t \in [n]} \Big).$$

Intuitively, monotonicity ensures that if more request predictions are correct, then the error must decrease. Lipschitzness ensures that the error measure can upper bound the difference between the optimal values of the actual instance and the predicted instance.

However, a natural extension of the error measure used in [18] exhibits a critical weakness—although it satisfies the two properties—that it labels poor predictions as good. This is because the extension fails to capture the requests' arrival time effectively; see Section II-B3 for the details. To address this challenge, we propose a novel error measure.

1) New Error Measure: Given an instance $I = (p_t)_{t \in [T]}$ and it prediction $\hat{I} = (\hat{p}_t)_{t \in [\hat{T}]}$, define $\mathsf{O}(I, \hat{I}) := (\max\{p_t, \hat{p}_t\})_{t \in [n]}$ and $\mathsf{U}(I, \hat{I}) := (\min\{p_t, \hat{p}_t\})_{t \in [n]}$ to be the *overpredicted* and *underpredicted* instances respectively. Assuming that we take the max and min at every time step, we can write $\mathsf{O}(I, \hat{I}) = \max\{I, \hat{I}\}$ and $\mathsf{U}(I, \hat{I}) = \min\{I, \hat{I}\}$.

Let $I\langle t_1, t_2 \rangle$ be the subinstance of I from time t_1 to t_2, i.e., $(p_t)_{t \in \{ t_1, \ldots, t_2 \}}$. Define $\mathcal{P} := \{ L_1 = \{ l_0, \ldots, l_1 \}, L_2 = \{ l_1 + 1, \ldots, l_2 \}, \cdots, \}$ to be a partition of the integer set $[T]$. The set of consecutive time steps, L_i, is called an *interval*. A partition $\mathcal{P}$ is called *non-empty* for instance I if and only if $I\langle L_i \rangle$ includes at least one request for every $L_i \in \mathcal{P}$. Let $\prod(I)$ be the set of all non-empty partitions for instance I. Note that different partitions may have a different number of intervals. We are now ready to define our error measure.

Definition 2. *(Error Measure) Given an instance* $I = (p_t)_{t \in [T]}$ *and its predicted instance* $\hat{I} = (\hat{p}_t)_{t \in [\hat{T}]}$*, the error measure is defined as follows:*

$$\eta(I, \hat{I}) = \max_{\mathcal{P} \in \prod(\mathsf{U}(I, \hat{I}))} \sum_{L_i \in \mathcal{P}} \Big(\mathrm{OPT}\Big(\mathsf{O}(I\langle L_i \rangle, \hat{I}\langle L_i \rangle) \Big) - \mathrm{OPT}\Big(\mathsf{U}(I\langle L_i \rangle, \hat{I}\langle L_i \rangle) \Big) \Big)$$

To understand η, for a moment, assume that $\mathcal{P}$ has only one interval. Then, η measures how much the optimal objective changes when the number of requests increases from $\min\{p_t, \hat{p}_t\}$ to $\max\{p_t, \hat{p}_t\}$ at all times. Although it satisfies Monotonicity and Lipschitzness, it fails to capture how requests arriving at different times affect the ack times—if they change the ack times significantly, intuitively, the prediction is not so good. Thus, we partition the time horizon $[T]$ into intervals and apply the same measure to each interval in $\mathcal{P}$. Intuitively, if the error is big for some partition, it means I and $\hat{I}$ have significantly different optimal solution structures.

When the parameters are clear in the context, we may write $\eta(I, \hat{I})$ as η for brevity. We claim the following lemma.

Lemma 1. *$\eta(I, \hat{I})$ can be efficiently computed by dynamic programming and satisfies both monotonicity and Lipschitzness.*

2) Auxiliary Prediction Error: For the sake of analysis, we define another error measure called *Auxiliary Error*.

Definition 3. *(Auxiliary Error) Given* $I = (p_t)_{t \in [T]}$*,* $\hat{I} = (\hat{p}_t)_{t \in [\hat{T}]}$*, the auxiliary error in time interval* $[t_1, t_2]$ *is defined as follows:*

$$\tau\Big([t_1, t_2], I, \hat{I} \Big) = \mathrm{OPT}\Big(\mathsf{O}(I\langle t_1, t_2 \rangle, \hat{I}\langle t_1, t_2 \rangle) \Big) - \mathrm{OPT}\Big(\mathsf{U}(I\langle t_1, t_2 \rangle, \hat{I}\langle t_1, t_2 \rangle) \Big)$$

Note that $\eta(I,\hat{I}) = \max_{\mathcal{P}\in\prod(U(I,\hat{I}))} \sum_{L_i\in\mathcal{P}} \Big(\tau(L_i, I, \hat{I})\Big)$. When the parameters are clear from the context, we may write $\tau([1,n], I, \hat{I})$ as τ for simplicity, where $n = \max\{T, \hat{T}\}$. The auxiliary error can be used to lower bound η and will be useful for the analysis.

Lemma 2. *For any instance I and its prediction $\hat{I}$ and for any non-empty partition $\mathcal{P} = \{L_1, L_2, \dots\}$ of $\mathsf{U}(I,\hat{I})$, we have $\sum_{L_i\in\mathcal{P}} \Big(\tau(L_i, I, \hat{I})\Big) \le \eta(I,\hat{I})$. Moreover, the axiliary error $\tau([1,n], I, \hat{I})$ satisfies Monotonicity and Lipschitzness.*

3) Comparison with Other Prediction Errors: Below we compare our error measure with other natural measures and discuss their shortcomings.

$\mathsf{ERR}(I,\hat{I}) = |\mathrm{OPT}(I) - \mathrm{OPT}(\hat{I})|$. This definition measures the difference of the optimal objectives of I and $\hat{I}$. It satisfies Lipschitzness but violates the Monotonicity: Consider instance $I = (2d, 2d, 0)$ and its predicted instance $\hat{I} = (0, 2d, 2d)$. Clearly, $|\mathrm{OPT}(I) - \mathrm{OPT}(\hat{I})| = 0$, as $\mathrm{OPT}(I) = \mathrm{OPT}(\hat{I}) = 2$. However, if we replace $\hat{p}_1$ with p_1 to get a more accurate prediction $(2d, 2d, 2d)$, $\mathsf{ERR}(I,\hat{I})$ increases to 1 from 0.

$\mathsf{ERR}(I,\hat{I}) = \sum_{t\in[T]} |p_t - \hat{p}_t|$. This definition linearly aggregates the difference of the actual number of requests from prediction over all times. This ℓ_1-norm measure satisfies Monotonicity but violates Lipschitzness: Consider instance $I = (p_t = \epsilon)_{t\in[T]}$ where $\frac{\epsilon n(n-1)}{2d} < 1$; thus the optimal solution only acks at the last time. The prediction is $\hat{I} = (\hat{p}_t)_{t\in[T]}$ where $\hat{p}_t = 0$ for all $t \in [T-1]$ and $\hat{p}_T = \epsilon$. Clearly, the optimal solution for $\hat{I}$ is the same as that for I but it incurs no delay cost for $\hat{I}$. In this case, we have $|\mathrm{OPT}(I) - \mathrm{OPT}(\hat{I})| = \frac{n(n-1)\epsilon}{2d}$. But, $\mathsf{ERR}(I,\hat{I}) = (n-1)\epsilon$, so we have $|\mathrm{OPT}(I) - \mathrm{OPT}(\hat{I})| \ge \frac{n}{2d}\mathsf{ERR}(I,\hat{I})$. Thus, we need to add $\Theta(n)$ multiplier to the measure to make it usable.

$\mathsf{ERR}(I,\hat{I}) = \tau([1,n], I, \hat{I})$. This error measures how much the optimal objective changes when increasing $\min\{p_t, \hat{p}_t\}$ to $\max\{p_t, \hat{p}_t\}$ at all times. This error measure satisfies both Monotonicity and Lipschitzness but is short of capturing the structure of the optimal solution. In particular, it may tag poor predictions as good. To see this, consider an instance I where a cluster of requests arrive initially until time t_1 and one request arrives at a very late time t_2. Then, we can set parameters appropriately, so the optimal solution makes only one ack at time t_2 and has cost $2-\epsilon$. Suppose the prediction $\hat{I}$ is perfect except at time t_1, and $\hat{p}_{t_1}$ is very large. Then, it is easy to see that acking at both times t_1 and t_2 is optimal and has cost 2. Thus, the error is at most ϵ, yet I and $\hat{I}$ have very different optimal solution structures. We can amplify their structural difference by repeating this example over time.

III. Consistency Bound

In this section, we present an algorithm and prove the consistency. More precisely, we show that for any instance I, the algorithm always returns a solution with the cost at most $(1+\epsilon)\cdot\mathrm{OPT}(I) + O(\frac{1}{\epsilon})\cdot\eta$ for any $\epsilon > 0$. When the prediction error $\eta = 0$, the competitive ratio is $(1+\epsilon)$. Later in Section IV, we show how to refine the algorithm to obtain robustness simultaneously. We begin by introducing a crucial definition that is necessary to understand our algorithmic intuition.

A. Stability of Instances

Let Y be a feasible solution of an instance $I = (p_t)_{t\in[T]}$, i.e., the set of ack time points. Use $\mathrm{D}(I,Y)$ to denote the total delay cost of solution Y and define $\Delta(I,Y,t) := \mathrm{D}(I,Y) - \mathrm{D}(I, Y\cup\{t\})$ to be the decrease of the delay cost when making an extra ack at time t in addition to an existing solution Y.

Definition 4. *For a parameter λ, let an instance $I = (p_t)_{t\in[T]}$ be a λ-stable interval if the solution X which sends only one ack at time T has $\Delta(I,X,t) \le 1-\lambda$, $\forall t \in [T]$. Define an instance's stability factor as the maximum value of λ such that it is a λ-stable interval. Further, the instance is stable if its stability factor is at least 0; otherwise, it is unstable.*

Clearly, any instance where the optimal solution X sends only one ack at time T is at least 0-stable because for any time t, $\Delta(I,X,t) \le 1$; otherwise, making an extra ack at time t decreases the total cost. Suppose that the stability factor of the instance is exactly 0. Namely, there exists at least one time t such that $\Delta(I,X,t) = 1$. We see that increasing p_t slightly will force the optimal solution to make an extra ack. Thus, for a stable interval, the stability factor λ measures how much noise can change the structure of the optimal solution on it.

Defining stability is critical for our algorithm. Intuitively, when the predicted instance is a stable interval, it is convenient to detect a considerable prediction and justify the extra acks. Thus, if we can partition the predicted instance into several stable intervals without incurring too much cost, handling each stable interval independently gives a consistent algorithm.

In the following, Section III-B presents an algorithm to deal with the case that the predicted instance is a stable interval, and then Section III-C shows how to do partitioning in the general case and obtain a desirable competitive ratio.

B. Stable Prediction Case

This subsection considers a special case that the predicted instance is a λ-stable interval, i.e., the optimal solution $\hat{X}$ only sends one ack, and $\Delta(\hat{I},\hat{X},t) \le 1-\lambda$ for all $t \in [\hat{T}]$. The algorithm is stated in Algorithm 1. It consists of two phases. The first phase is concerned with "good" prediction cases, while the second phase runs a traditional competitive algorithm to handle inputs that turn out to be far off from the prediction. The pseudo-code uses the 2-competitive algorithm that is deterministic, but it can also be replaced with the $\frac{e}{e-1}$ competitive algorithm that is randomized.

To decide if the predictions are reliable or not, the algorithm uses $(1+\lambda)\mathrm{OPT}(\hat{I})$ as "budget". The algorithm does not switch to the second phase if it has paid within the budget. For each time t in the first phase, the algorithm acks if the subinstance $I\langle j+1, t+1\rangle$ is unstable, where j is the last time an ack was made. Using the fact that the prediction makes only

Algorithm 1. Predicted-Budget-Based Algorithm

Input: Online Instance $I = (p_t)_{t\in[T]}$, prediction $\hat{I} = (\hat{p}_t)_{t\in[\hat{T}]}$, and parameter $\lambda > 0$
Output: A feasible solution S

$S \leftarrow \emptyset$; $j \leftarrow 0$; $t \leftarrow 1$.
Compute $\mathrm{OPT}(\hat{I})$ for the predicted instance.
// Phase-1.
while $\mathrm{F}(I\langle 1, t\rangle, S\cup\{\,t\,\}) < (1+\lambda)\mathrm{OPT}(\hat{I})$ and the online instance does not end **do**
  if $I\langle j+1, t+1\rangle$ is unstable **then**
    $S \leftarrow S \cup \{\,t\,\}$. // Send an ack at time point t.
    $j \leftarrow t$.
  end if
  $t \leftarrow t+1$.
end while
// Phase-2.
if the online instance does not end **then**
  Run the traditional 2-competitive online algorithm for the remaining instance and let Y be the returned solution. Let $S \leftarrow S \cup Y$.
end if
return S

one ack at the end of a single interval in a stable instance, we can show that the prediction has a considerable error. Thus, we can charge the cost for making extra acks to the error. Also, the algorithm makes an ack when it is forced to finish the first phase due to running out of budget. It is worth noting that the switched time point may be larger than $\hat{T}$, which is the last time point in the predicted instance.

Theorem 2. *Let* $\mathrm{ALG}(I)$ *be the objective value obtained by Algorithm 1. If the predicted instance is a* λ*-stable interval* ($\lambda > 0$)*,* $\mathrm{ALG}(I) \le (1+\lambda)\cdot\mathrm{OPT}(\hat{I}) + O(\frac{1}{\lambda})\cdot\tau$*, where* $\tau = \tau([1,n], I, \hat{I})$ *is the auxiliary error (see Definition 3), which is a lower bound of the prediction error* η*.*

We sketch the proof of Theorem 2. Use X to denote the solution returned by Algorithm 1. As stated in Algorithm 1, the algorithm consists of two phases. Let e be the last time in the first phase; then $e \in X$ by the algorithm. Time e partitions $[T]$ into two parts: $S_a = \{\,1,\ldots,e\,\}$ and $S_b = \{\,e+1,\ldots,T\,\}$. It is worth to note that e may equal T, which makes $S_b = \emptyset$. Define $X_a := \{\,t \in X \mid t \le e\,\}$, $X_b := X\backslash X_a$, $I_a := (p_i)_{i\in S_a}$ and $I_b := (p_i)_{i\in S_b}$. Then, we can split the objective value into two parts: $\mathrm{ALG}(I) = \mathrm{F}(I_a, X_a) + \mathrm{F}(I_b, X_b)$, where the first and second terms are the cost incurred for the algorithm in the first and second phases, respectively. Theorem 2 can be proved by the following two lemmas.

Lemma 3. $\mathrm{F}(I_a, X_a) \le (1+\lambda)\cdot(\mathrm{OPT}(I) + \tau)$.

Proof. This lemma is easy to prove because the first phase is budget limited, that is, all cost incurred in this phase is at most the budget $(1+\lambda)\mathrm{OPT}(\hat{I})$. Then due to the Lipschitzness of the auxiliary error (Lemma 2), we have $\mathrm{OPT}(\hat{I}) \le \mathrm{OPT}(I) + \tau$, completing the proof. □

Lemma 4. $\mathrm{F}(I_b, X_b) \le 2\cdot\mathrm{OPT}(I_b) \le (2+\frac{4}{\lambda})\cdot\tau$.

Proving Lemma 4 is a bit subtle. Before describing it, we first state a claim critical to the analysis.

Claim 1. *Consider an arbitrary instance* I*. Partition it into* k *intervals (subinstances)* $\{I_1, I_2, \ldots, I_k\}$*. For any such partition, We have* $\sum_{i\in[k]}\mathrm{OPT}(I_i) \le k - 1 + \mathrm{OPT}(I)$.

Proof. Let t_i be the last time point of subinstance I_i. We, w.l.o.g., assume that the optimal solution X^* sends an ack at the last time point, implying that $t_k \in X^*$. Construct a new solution $Y := X^* \cup \{t_1, t_2, \ldots, t_{k-1}\}$. Clearly, $\mathrm{F}(I, Y) \le k - 1 + \mathrm{OPT}(I)$ because the number of acks increases at most $k-1$ and the total delay cost is non-increasing.

Now partition solution Y into k groups $Y_1, \ldots, Y_k$, where Y_i is the set of the ack time points in the time interval of I_i. Since $t_i \in Y$ for any $i \in [k]$, we have $t_i \in Y_i$, and thus, the objective value of solution Y is exactly the sum of its values in k subinstances, i.e., $\mathrm{F}(I, Y) = \sum_{i=1}^{k}\mathrm{F}(I_i, Y_i)$. Then due to $\mathrm{F}(I_i, Y_i) \ge \mathrm{OPT}(I_i)$ for any $i \in [k]$, we prove the claim. □

Proof of Lemma 4. The first inequality uses the fact that the second phase runs the traditional 2-competitive algorithm. So we only need to focus on the second inequality. We distinguish two cases according to the existence of phase-2. The first case is that the algorithm is still in phase-1 at time T, i.e., $I_b = \emptyset$. Then the second inequality is trivial since $\mathrm{OPT}(I_b) = 0$. For the second case that $I_b \neq \emptyset$, since $\{I_a, I_b\}$ is a partition of instance I, by Claim 1, $\mathrm{OPT}(I_b) \le 1 + \mathrm{OPT}(I) - \mathrm{OPT}(I_a)$.

In phase-1, according to the "if-condition" in Algorithm 1, the subinstance between any two adjacent acks is a stable interval, implying that the algorithm obtains the optimal solution for each subinstance. Thus, using Claim 1 again gives a lower bound of $\mathrm{OPT}(I_a)$: $\mathrm{OPT}(I_a) \ge \mathrm{F}(I_a, X_a) - k_a + 1$, where k_a is the number of acks in phase-1. Since $I_b \neq \emptyset$, the first phase must run out of the predicted budget, which indicates that $\mathrm{F}(I_a, X_a) \approx (1+\lambda)\mathrm{OPT}(\hat{I})$. Without loss of generality, we can assume $\mathrm{F}(I_a, X_a) \ge \mathrm{OPT}(\hat{I})$. Hence, $\mathrm{OPT}(I_b) \le 1 + \mathrm{OPT}(I) - \mathrm{OPT}(\hat{I}) + k_a - 1 \le \tau + k_a$, where the last inequality is due to the Lipschitzness of τ.

The remaining piece of proof is to bound the value of k_a. Up to this point, we have not used the condition that the predicted instance $\hat{I}$ is a λ-stable interval. The following analysis uses this condition to prove $k_a \le 2\tau/\lambda$.

Use $\{I_1, \ldots, I_{k_a}, \ldots, I_k\}$ to denote the subinstances induced by solution X. Let $T_i = [s_i, t_i]$ be the time interval of subinstance I_i. Recall that $\mathsf{O}(I, \hat{I})$ is the over-predicted instance $(\max\{p_t, \hat{p}_t\})_{t\in n}$, where $n = \max\{T, \hat{T}\}$. For brevity, denote $\mathsf{O}(I, \hat{I})$ by $I^O = (p_t^O)_{t\in n}$, where $p_t^O = \max\{p_t, \hat{p}_t\}$. Let $\hat{X}$ be the optimal solution of the predicted instance $\hat{I}$. Since we assume that $\hat{I}$ is λ-stable, $\hat{X} = \{\hat{T}\}$.

Claim 2. *The optimal solution of instance* I^O *sends at least one ack in interval* T_i *for any* $i \in [k_a]$.

Claim 3. *For any interval partition $\{\hat{I}_1, \ldots, \hat{I}_l\}$ of instance $\hat{I}$, use $Y_i = \{y_i\}$ to denote the solution that only sends an ack at the last time point of $\hat{I}_i$. We have* $\mathrm{D}(\hat{I}, \hat{X}) \leq \sum_{i=1}^{l} \mathrm{D}(\hat{I}_i, Y_i) + \sum_{i=1}^{l} \Delta(\hat{I}, \hat{X}, y_i)$.

We directly use the above two claims and defer their proofs to the full version of this paper. Use solution $Y = \{y_1, \ldots, y_l\}$ to denote the optimal solution of instance I^O. By Claim 2, we know that solution Y sends at least k_a acks, i.e.,

$$l \geq k_a. \tag{3}$$

Thus, we turn to upper bound the value of l. Notice that

$$l = \mathrm{OPT}(I^O) - \mathrm{D}(I^O, Y), \tag{4}$$

Use $\{I_1^O, \ldots, I_l^O\}$ to denote the subinstances of instance I^O induced by solution Y. Then, $\mathrm{D}(I^O, Y) = \sum_{i=1}^{l} \mathrm{D}(I_i^O, \{y_i\})$. We further define $\hat{I}_i$ to be the subinstance of $\hat{I}$ which shares the same time interval of I_i^O. Clearly,

$$\mathrm{D}(I^O, Y) = \sum_{i=1}^{l} \mathrm{D}(I_i^O, \{y_i\}) \geq \sum_{i=1}^{l} \mathrm{D}(\hat{I}_i, \{y_i\}). \tag{5}$$

By Claim 3, we can connect the above quantity to $\mathrm{OPT}(\hat{I})$:

$$\sum_{i=1}^{l} \mathrm{D}(\hat{I}_i, \{y_i\}) \geq \mathrm{D}(\hat{I}, \hat{X}) - \sum_{i=1}^{l} \Delta(\hat{I}, \hat{X}, y_i). \tag{6}$$

Due to the assumption that $\hat{I}$ is a λ-stable interval and the fact that $\Delta(\hat{I}, \hat{X}, y_l) = 0$,

$$\mathrm{D}(\hat{I}, \hat{X}) = \mathrm{OPT}(\hat{I}) - 1, \tag{7}$$

and

$$\sum_{i=1}^{l} \Delta(\hat{I}, \hat{X}, y_i) \leq (1 - \lambda) \cdot (l - 1). \tag{8}$$

Combining the above inequalities, we have

$$l + \sum_{i=1}^{l} \mathrm{D}(\hat{I}_i, \{y_i\}) \leq \mathrm{OPT}(I^O) \qquad \text{(Eq. (4) \& (5))}$$

$$l + \mathrm{D}(\hat{I}, \hat{X}) \leq \mathrm{OPT}(I^O) + \sum_{i=1}^{l} \Delta(\hat{I}, \hat{X}, y_i) \qquad \text{(Eq. (6))}$$

$$\lambda \cdot l - \lambda \leq \mathrm{OPT}(I^O) - \mathrm{OPT}(\hat{I}) \qquad \text{(Eq. (7) \& (8))}$$

$$k_a \leq \frac{\tau}{\lambda} + 1. \qquad \text{(Eq. (3) \& Def. 3)}$$

The last piece is to carefully show that τ/λ is always at least 1, which implies that $k_a \leq 2\tau/\lambda$ and completes the proof. If $k_a \geq 2$, we have $\tau/\lambda \geq k_a - 1 \geq 1$. Then if $k_a = 1$, the algorithm sends only one ack in phase-1. Thus, I_a is a stable interval and $F(I_a, X_a) = \mathrm{OPT}(I_a) < \mathrm{OPT}(I^O)$. Since the algorithm runs out of the predicted budget $(1 + \lambda)\mathrm{OPT}(\hat{I})$ and enters phase-2, we have $\mathrm{OPT}(I^O) \geq (1 + \lambda) \cdot \mathrm{OPT}(\hat{I})$. Again, due to Def. 3, $\tau \geq \lambda \cdot \mathrm{OPT}(\hat{I}) \geq \lambda$, indicating that τ/λ is still at least 1.

□

Proof of Theorem 2. Combining Lemma 3 and Lemma 4 directly proves the theorem. □

Algorithm 2. Adaptive Predicted-Budget-Based Algorithm

Input: Online Instance $I = (p_t)_{t \in [T]}$, prediction $\hat{I} = (\hat{p}_t)_{t \in [\hat{T}]}$ and parameter $\lambda > 0$.
Output: A feasible solution X.

```
X ← ∅; t' ← 0.
Let Y be a λ-stable 1/(1−λ)-approximation solution of Î.
while t' < T̂ and Î⟨t', T̂⟩ is not a λ-stable interval do
    Let t̂ be the minimum time point in Y which is larger than t'.
    Run Algorithm 1 on the input {I⟨t'+1, T⟩, Î⟨t'+1, t̂⟩, λ} until the first phase ends.
    Let Z be the returned solution and t'' be the termination time.
    if t'' < t̂ then
        Run the traditional 2-competitive algorithm for instance I⟨t''+1, t̂⟩ and let Y be the returned solution
    end if
    X ← X ∪ Z ∪ Y; t' ← max{t'', t̂}.
end while
if t' < T̂ and the online instance does not end then
    Run Algorithm 1 on the input {I⟨t'+1, T⟩, Î⟨t'+1, T̂⟩, λ} and let Z be the returned solution.
    X ← X ∪ Z.
end if
return X
```

C. General Algorithm

Now we consider the general case. As mentioned above, the basic idea is partitioning the instance into several stable prediction subinstances and dealing with each subinstance separately. The key challenge here is how to partition the instance such that the sum of the subinstances' optimal values is close to the optimal value of the original instance. We first give the following statements to show that it is not ridiculous that such a partition exists.

Definition 5. *For an instance I, a feasible solution Y with k acks partitions it into k subinstances $\{I_1, I_2, \ldots, I_k\}$. We say solution Y is λ-stable if any subinstance induced by it is a λ-stable interval, i.e., for any time t of the whole instance I, $\Delta(I, Y, t) \leq 1 - \lambda$.*

Lemma 5. *For any instance $I = (p_t)_{t \in [T]}$ of DAP and any $\lambda \in [0, 1)$, there exists a λ-stable solution which is $\frac{1}{1-\lambda}$-approximation and can be computed in polynomial time.*

The proof is technically simple and omitted here. The upshot is that we start from an optimal solution and always make an extra ack at time t where the Δ value is larger than $1 - \lambda$. Clearly, each extra ack increases the objective value by at most λ. Thus, the newly incurred cost is at most λ times the number of acks in the new solution, implying the total cost of the new solution can be bounded.

According to Lemma 5, we can easily split the predicted instance $\hat{I}$ into several λ-stable intervals and the incurred

cost is at most $O(\lambda)\mathrm{OPT}(\hat{I})$. Then due to the Lipschitzness of our prediction error (Lemma 1), the cost is at most $O(\lambda)(\mathrm{OPT}(I)+\eta)$.

The main algorithm is given in Algorithm 2. It iteratively treats an interval of λ-stable $O(\lambda)$-approximation solution as an instance of the stable prediction case and calls Algorithm 1.

From the description, we see that iterations are handled independently. In each iteration, the algorithm always starts by trusting the prediction regardless of the states in the previous iterations and enters the next iteration if (i) Algorithm 1 runs out of the predicted budget of the current iteration; and (ii) Algorithm 1 has processed requests until the last time point of the current predicted interval. Since the algorithm adapts in each iteration, we refer to it as an adaptive algorithm. We claim the following theorem.

Theorem 3. *For an instance $I = (p_t)_{t\in[T]}$ and its prediction $\hat{I} = (\hat{p}_t)_{t\in[\hat{T}]}$, let the $\mathrm{ALG}(I)$ be the cost of the solution returned by Algorithm 2 with $\lambda = \Theta(\epsilon)$, then we have:$\mathrm{ALG}(I) \le (1+\epsilon)\cdot\mathrm{OPT}(I) + O(\frac{1}{\epsilon})\cdot\eta$. Further, the running time is $O(n^2)$, where $n = \max\{T,\hat{T}\}$.*

The analysis follows from aggregating the bounds over all the intervals. We can easily show that $\sum_i(\mathrm{OPT}(\hat{I}_i)) \le (1 + O(\lambda))\cdot\mathrm{OPT}(\hat{I})$ and $\sum_i \tau_i \le \eta$, where $\hat{I}_i$ is the i-th stable prediction subinstance and τ_i is its auxiliary error. Then, using Theorem 2 and Lemma 5 proves the claimed competitive ratio in Theorem 3. For the running time, the most time-consuming part is computing the optimal offline solution of the predicted instance, while all other operations can be implemented in the linear time. Since the optimal solution can be computed by a $O(n^2)$ dynamic programming algorithm [17], the running time can be proved easily. The whole proof of Theorem 3 is straightforward. Due to space, we omit the details here.

D. Optimality of Consistency

We claim the following theorem to show that the dependence of the prediction error η in Algorithm 2 is almost the best possible. The detailed proof is deferred to the full version.

Theorem 4. *Given an instance I and its prediction $\hat{I}$, the solution of any deterministic algorithm is at least $\min\{(1+\lambda)\cdot\mathrm{OPT}(I), \mathrm{OPT}(I)+\frac{\eta}{\lambda}\}$, where $\lambda > 0$ is a parameter.*

IV. Robustness Bound

This section refines Algorithm 2 to obtain robustness bounds in addition to the consistency bound. Here we only discuss the high-level ideas. For each time t, define $I_t := I\langle 1,t\rangle$ and $\eta_t := \eta(I_t \cup \hat{I}\langle t+1,n\rangle, \hat{I})$. Due to the monotonicity of the error, η_t increases as t increases. An intuitive way to gain robustness is switching to the 2-competitive deterministic (or $e/(e-1)$-competitive randomized) algorithm when η_t is found to be large. If we know the optimal value $\mathrm{OPT}(I)$, we can then make the switch at the first time t we observe the error is large, i.e., $\eta_t > \epsilon\mathrm{OPT}(I)$. Due to the monotonicity discussed above, the actual error η will only be large. Thus, we can achieve $(1+\epsilon)$-consistency bound along with $(2+\epsilon)$-robustness bound (or $e/(e-1)+\epsilon$ if randomization is allowed).

However, we can only see I_t at each time t and thus, $\mathrm{OPT}(I)$ is unknown. The current error η_t could look large compared to $\mathrm{OPT}(I_t)$, but turn out to be very small compared to $\mathrm{OPT}(I)$. We address this issue based on the observation that an instance can be partitioned into several subinstances such that the optimal cost of each subinstance is at most $1/\epsilon$ while increasing the aggregate optimal cost by at most $(1+\epsilon)$ factor. Further, this partition can be done online by checking the optimal solution to the current subinstance at every time point. Then, for each subinstance, we can argue that if the current error is large enough to shift to the traditional algorithm, it is also large against the optimum for the subinstance. In this process, to obtain a robustness guarantee, we increase the coefficient of η from $O(\frac{1}{\epsilon})$ to $O(\frac{1}{\epsilon^2})$ in the consistency bound for a technical reason. Combing Theorem 3 and the robustness scheme, Theorem 1 can be proved.

V. Learnability

This section shows that we can learn a prediction with approximately minimum expected error given only a polynomial number of samples. We make standard assumptions that the number of time steps is at most T, the number of packages per time step is at most K, and each DAP instance is sampled from an unknown distribution $\mathcal{D}$. Use $\mathcal{I}$ to denote the set of all potential predictions $\hat{I}$, i.e., $\mathcal{I} := \{\,(p_1,p_2,\ldots,p_T) \mid \forall t\in[T], p_t\in[0,K]\,\}$. We claim the following theorem and defer the proof to the full version.

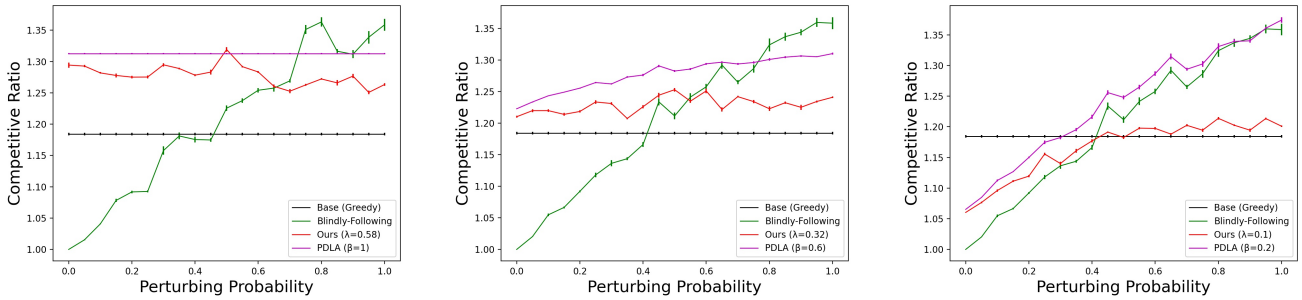
Theorem 5. *For any $\epsilon,\delta \in (0,1)$ and any distribution $\mathcal{D}$, after observing $O((\frac{T}{\epsilon})^2(T\log(\frac{KT}{\epsilon d}) + \ln(\frac{1}{\delta}))$ samples, there exists a learning algorithm that returns a predicted instance $\hat{I}\in\mathcal{I}$ such that with probability at least $1-\delta$, $\mathbb{E}_{I\sim\mathcal{D}}[\eta(\hat{I},I)] \le \mathbb{E}_{I\sim\mathcal{D}}[\eta(I^*,I)] + \epsilon$, where for any two instances I_1, I_2, $\eta(I_1,I_2)$ represents the error when I_1 is the prediction of I_2, and $I^* = \arg\min_{I'\in\mathcal{I}}\mathbb{E}_{I\sim\mathcal{D}}[\eta(I',I)]$.*

VI. Experiments

This section empirically validates our adaptive learning-augmented algorithm (ALA) 's efficiency. We investigate various types of the input distribution and show superior performance compared to previous algorithms.

A. Setup

a) Input distributions and Noisy Predictions: The experiments follow the setting in [9]. We set the delay factor $d = 100$ and the maximum number of time steps $T = 1000$. For each time point, the number of demands is i.i.d. sampled from a given distribution $\mathcal{D}$. We investigate the same distributions as in [9]: the Poisson distribution of mean 1, the Pareto distribution of shape 2 and the iterated Poisson distribution of mean 1. The iterated Poisson distribution is a custom distribution introduced in [9], which is iterating on sampling a value from the Poisson distribution whose mean is the sampled value in the last iteration (initially, the mean is 1). The prediction of an instance is constructed by perturbing it with some noise. For each time t, there are two operations: setting



(a) $\beta = 1, \lambda = 0.58$ (b) $\beta = 0.6, \lambda = 0.32$ (c) $\beta = 0.2, \lambda = 0.1$

Fig. 1: The performance of the algorithms under the Pareto distribution. A large perturbing probability implies a large prediction error. Recollect that PDLA [9] parameterized by β has a consistency ratio of $\frac{\beta}{1-e^{-\beta}}$, and our algorithm parameterized by λ a consistency ratio of $(1+\lambda)$. For fair comparison, we consider different pairs of (β, λ) that give the same consistency ratio for PDLA and our algorithm.

$p_t = 0$ and adding to p_t a random noise sampled from $\mathcal{D}$. We perform each operation sequentially and independently with probability $r \in [0, 1]$ at each time point. Then, the perturbed instance is served as the prediction. The perturbing probability can be viewed as a simplified measure of the prediction error. The experiments test the performance of algorithms over different perturbing probabilities under each input distribution.

b) Baseline Algorithms: In addition to our algorithm, which is parameterized by λ, we implemented the following algorithms for comparison:

- GREEDY [17]. This algorithm acks when the cumulative delay cost equals 1 (the ack cost). It is the best deterministic algorithm without predictions.
- PDLA [9]. This algorithm is the first algorithm incorporating predictions. The algorithm has a control parameter $\beta \in (0, 1]$, and has a better consistency guarantee with a smaller value of β at the cost of a worse robustness guarantee. This was termed as the primal dual learning-augmented algorithm by the authors.
- BLINDFOLLOWING. This algorithm follows the prediction blindly. It applies the optimal solution on the predicted instance to the actual instance with no adaptation.

Note that we did not include the $\frac{e}{e-1}$-competitive randomized algorithm as it rarely outperforms the 2-competitive deterministic algorithm in practice.

c) Computational Settings.: We conduct experiments on a machine running Ubuntu 18.04 with an i7-7800X CPU and 48 GB memory. The algorithms are implemented in Python 3.8, and the results are averaged over five runs.

B. Empirical Discussion

All the results for the three distributions considered exhibiting similar patterns. Thus, we show the results only for the Pareto distribution in Fig. 1, deferring the other results to the full version of this paper. We observe the following.

- Our algorithm is robust to the error. Further, with a small λ, our algorithm has a better empirical competitive ratio than algorithm GREEDY when the prediction error (perturbing probability) is small and remains on par with it, regardless of the error.
- For the choice of β and λ values that lead to the same consistency guarantee, our algorithm shows a better performance than PDLA in most cases. This demonstrates that our algorithm obtains a better trade-off between consistency and robustness, confirming the importance of simultaneous optimum consistency and robustness achieved by our algorithm.

VII. CONCLUSION

This paper revisited the dynamic acknowledgment problem. For this problem, previously, it was unclear what a good error measure should be in the learning augmented algorithm analysis model. One of this paper's main contributions lies in formulating a novel error measure and designing algorithms based on the error. The algorithm developed in this paper achieves simultaneous optimum consistency and robustness, the most desirable result. The theory is verified empirically. We believe our new error and algorithm could inspire new ML-augmented solutions for other problems with a temporal nature to their input[5].

[5]The authors have provided public access to their code at https://github.com/Chenyang-1995/TCP

REFERENCES

[1] A. Al-Jubari, M. Othman, B. M. Ali, and N. A. W. A. Hamid, "An adaptive delayed acknowledgment strategy to improve TCP performance in multi-hop wireless networks," *Wirel. Pers. Commun.*, vol. 69, no. 1, pp. 307–333, 2013.

[2] K. Anand, R. Ge, and D. Panigrahi, "Customizing ML predictions for online algorithms," in *ICML*, 2020, pp. 303–313.

[3] A. Antoniadis, C. Coester, M. Elias, A. Polak, and B. Simon, "Online metric algorithms with untrusted predictions," in *ICML*, 2020, pp. 345–355.

[4] A. Antoniadis, T. Gouleakis, P. Kleer, and P. Kolev, "Secretary and online matching problems with machine learned advice," in *NeurIPS*, 2020.

[5] F. R. Armaghani, S. S. Jamuar, S. Khatun, and M. F. A. Rasid, "Performance analysis of TCP with delayed acknowledgments in multi-hop ad-hoc networks," *Wirel. Pers. Commun.*, vol. 56, no. 4, pp. 791–811, 2011.

[6] Y. Azar, S. Leonardi, and N. Touitou, "Flow time scheduling with uncertain processing time," in *STOC*. ACM, 2021, pp. 1070–1080.

[7] Y. Azar and N. Touitou, "Beyond tree embeddings - a deterministic framework for network design with deadlines or delay," in *FOCS*. IEEE, 2020, pp. 1368–1379.

[8] É. Bamas, A. Maggiori, L. Rohwedder, and O. Svensson, "Learning augmented energy minimization via speed scaling," in *NeurIPS*, 2020.

[9] É. Bamas, A. Maggiori, and O. Svensson, "The primal-dual method for learning augmented algorithms," in *NeurIPS*, 2020.

[10] C. Brito, E. Koutsoupias, and S. Vaya, "Competitive analysis of organization networks or multicast acknowledgement: how much to wait?" in *SODA*. SIAM, 2004, pp. 627–635.

[11] E. Brosh and Y. Shavitt, "Approximation and heuristic algorithms for minimum-delay application layer multicast trees," in *INFOCOM*. IEEE, 2004, pp. 2697–2707.

[12] N. Buchbinder, T. Kimbrel, R. Levi, K. Makarychev, and M. Sviridenko, "Online make-to-order joint replenishment model: Primal-dual competitive algorithms," *Oper. Res.*, vol. 61, no. 4, pp. 1014–1029, 2013.

[13] N. Buchbinder and J. Naor, "The design of competitive online algorithms via a primal-dual approach," *Found. Trends Theor. Comput. Sci.*, vol. 3, no. 2-3, pp. 93–263, 2009.

[14] R. Cheng and H. Lin, "Improving TCP performance with bandwidth estimation and selective negative acknowledgment in wireless networks," *J. Commun. Networks*, vol. 9, no. 3, pp. 236–246, 2007.

[15] Y. Daldoul, T. Ahmed, and D. Meddour, "IEEE 802.11n aggregation performance study for the multicast," in *Wireless Days*. IEEE, 2011, pp. 1–6.

[16] R. de Oliveira and T. Braun, "A dynamic adaptive acknowledgment strategy for TCP over multihop wireless networks," in *INFOCOM*. IEEE, 2005, pp. 1863–1874.

[17] D. R. Dooly, S. A. Goldman, and S. D. Scott, "TCP dynamic acknowledgment delay: Theory and practice (extended abstract)," in *STOC*. ACM, 1998, pp. 389–398.

[18] S. Im, R. Kumar, M. M. Qaem, and M. Purohit, "Non-clairvoyant scheduling with predictions," in *SPAA*. ACM, 2021, pp. 285–294.

[19] Z. Jiang, D. Panigrahi, and K. Sun, "Online algorithms for weighted paging with predictions," *ICALP*, pp. 69:1–69:18, 2020.

[20] A. R. Karlin, C. Kenyon, and D. Randall, "Dynamic TCP acknowledgement and other stories about e/(e-1)," in *STOC*. ACM, 2001, pp. 502–509.

[21] T. Kraska, A. Beutel, E. H. Chi, J. Dean, and N. Polyzotis, "The case for learned index structures," in *SIGMOD*, 2008, pp. 489–504.

[22] R. Kumar, M. Purohit, and Z. Svitkina, "Improving online algorithms using ML predictions," in *NeurIPS*, 2018, pp. 9661–9670.

[23] S. Lattanzi, T. Lavastida, B. Moseley, and S. Vassilvitskii, "Online scheduling via learned weights," in *SODA*, 2020, pp. 1859–1877.

[24] S. Li and J. Xian, "Online unrelated machine load balancing with predictions revisited," in *International Conference on Machine Learning*. PMLR, 2021, pp. 6523–6532.

[25] Z. Lu, W. Wu, W. W. Li, and M. Pan, "Efficient scheduling algorithms for on-demand wireless data broadcast," in *INFOCOM*. IEEE, 2016, pp. 1–9.

[26] T. Lykouris and S. Vassilvtiskii, "Competitive caching with machine learned advice," in *ICML*, 2018, pp. 3296–3305.

[27] M. Mahdian, H. Nazerzadeh, and A. Saberi, "Online optimization with uncertain information," *TALG*, vol. 8, no. 1, pp. 1–29, 2012.

[28] M. Mitzenmacher, "Scheduling with predictions and the price of misprediction," in *ITCS*, 2020, pp. 14:1–14:18.

[29] K. Mokhtarian and H. Jacobsen, "Minimum-delay multicast algorithms for mesh overlays," *IEEE/ACM Trans. Netw.*, vol. 23, no. 3, pp. 973–986, 2015.

[30] M. Purohit, Z. Svitkina, and R. Kumar, "Improving online algorithms via ml predictions," *Advances in Neural Information Processing Systems*, vol. 31, pp. 9661–9670, 2018.

[31] D. Rohatgi, "Near-optimal bounds for online caching with machine learned advice," in *SODA*, 2020, pp. 1834–1845.

[32] S. S. Seiden, "A guessing game and randomized online algorithms," in *STOC*. ACM, 2000, pp. 592–601.

[33] C. Su and L. Tassiulas, "Broadcast scheduling for information distribution," in *INFOCOM*. IEEE Computer Society, 1997, pp. 109–117.

[34] A. Wei, "Better and simpler learning-augmented online caching," in *APPROX-RANDOM*, ser. LIPIcs, vol. 176, 2020, pp. 60:1–60:17.