Online Learning Approaches in Maximizing Weighted Throughput

Zhi Zhang, Fei Li, Songqing Chen
Department of Computer Science
George Mason University
Fairfax, Virginia 22030
Email: {zzhang8, lifei, sqchen}@cs.gmu.edu

***Abstract*—Motivated by providing quality-of-service for next generation IP-based networks, we design algorithms to schedule packets with values and deadlines. Packets arrive over time; each packet has a non-negative value and an integer deadline. In each time step, at most one packet can be sent. Packets can be dropped at any time before they are sent. The objective is to maximize the total value gained by delivering packets no later than their respective deadlines. This model is the well-studied *bounded-delay model* (Hajek. CISS 2001. Kesselman et al. SICOMP 2004) which extensive competitive online algorithms have been developed for. In a generalization of this model, the success of delivering a packets in each time step depends on the reliability of the communication channel.**

In this paper, we apply online learning approaches on this model as well as a few of its variants. We design online learning algorithms and analyze their performance theoretically in terms of external regret. We also measure these algorithms' performance experimentally. We conclude that no online learning algorithms have a constant regret. Our online learning algorithms outperform the competitive algorithms for algorithmic simplicity and running complexity. However, in general, this online learning algorithms work no worse than the best known competitive online algorithm for maximizing weighted throughput in practice.

I. Introduction

In the past a few decades, the Internet continues serving various types of applications. The routers in the IP-based infrastructure execute the Internet's main functionality. Figure 1 illustrates the functionalities of the buffer management inside of routers. A buffer management algorithm is in charge of two tasks, *queuing packets* and *delivering packets*. When new packets arrive, buffer management decides which ones to accept and queue for potential deliveries, and which ones already in the buffer to drop permanently due to packet deadline or buffer capacity constraints. In each time step, the buffer management selects a pending packet in the buffer to send. Information about the incoming packets in the future is completely unknown to us. The buffer management can be regarded as an online scheduling algorithm processing prioritized packets.

For ease of implementation and deployment, most current routers forward packets in a first-in-first-out manner and treat all packets equally: They simply send the earliest-released packet and when the buffer is full of packets, later arrivals will be dropped immediately. However, the diversity of applications running over the Internet has resulted in heterogeneity, and unpredictable or even chaotic network traffic [23], [24]. Thus, it is more reasonable to consider differentiation among packets from different types of applications. One such effort is to provide the proportional differentiated services called the DiffServ (see [7], [19] and the references therein). These practical concerns have made buffer management at routers significant and challenging in providing effective quality-of-service (QoS) support to various applications.

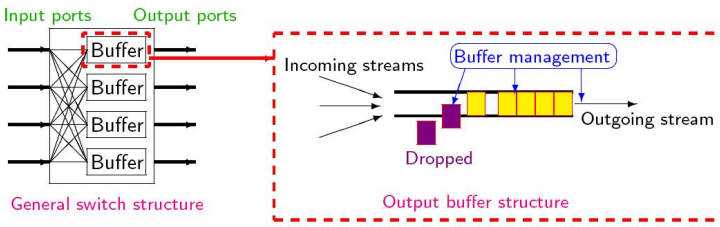

Fig. 1. Buffer management inside of routers

In the QoS buffer management setting, we use a *value* to represent a packet's priority, and we maximize the *weighted throughput*, which is defined as the total value of the packets sent. (In this paper, we use 'value', 'weight', and 'priority' interchangeably.) If the release time and value of each packet are known ahead of time, an optimal schedule can be found efficiently (see Section III-A). The optimal offline algorithm is empowered of clairvoyance to have the whole input sequence in advance to design its scheduling policy. However, in real applications, we do not know all such information ahead of time. Rather, packets arrive over time in an *online* manner, and we only learn about a packet and its associated characteristics when it actually arrives. Thus, an optimal offline scheduling algorithm is unrealistic and cannot be applied in real buffer management. Instead, we have to design *online buffer management* that makes its decision based on the input that it has seen so far.

A. Related work

Principally, there are two main streams of research targeting at providing differentiated services for the Internet via efficient buffer management. One kind of such research is to provide statistical guarantees under some mild assumptions on the stochastic properties of the traffic flows. These algorithms are called *stochastic algorithms*. Statistical guarantees heavily

978-1-4244-9328-9/10/$26.00 ©2010 IEEE

depend on the successfulness of modeling the input traffic flows and probabilistic analysis is used to calculate such an algorithm's expected performance.

In contrast to providing statistical guarantees, other researchers study the algorithms' performance under the worst case scenarios. These algorithms can be classified by *competitive online algorithms* and *online learning algorithms*. Quite extensive work has been done on competitive online algorithms. To our knowledge, our work in this paper is the first endeavor to tackle with this buffer management problem using the online learning approach.

Research on competitive online algorithms starts from Aiello et al. [1], Mansour et al. [22], and Kesselman et al. [12]. In their models, no assumptions are made on the input traffic flows and *competitive ratio* [4] is used to measure an online algorithm's performance. These studies avoid the difficulty introduced by not-successful-yet modeling of the input traffic flow [24]. The models in [1], [12] have attracted much attention and have been studied extensively in the past a few years. In order to evaluate the performance of an online competitive algorithm lacking of future input information, we compare it with an optimal offline algorithm. An online algorithm is called k-*competitive* if its weighted throughput on *any* instance is at least $1/k$ of the weighted throughput of an optimal offline algorithm on this instance. The *upper bounds* of competitive ratios are achieved by some known online algorithms. A competitive ratio less than the *lower bound* is not reachable by any online algorithm.

For the bounded-delay model (the buffer size is assumed ∞), an optimal offline algorithm has been proposed in [12], running in $O(n \log n)$ time where n is the number of packets released. For online algorithms, the best known lower bound of competitive ratio of deterministic algorithms is $\phi = \frac{1+\sqrt{5}}{2} \approx 1.618$ [11], [6], [2]; this lower-bound also applies to instances in which the deadlines of the packets (weakly) increase with their release dates. A simple greedy algorithm that always schedules the maximum-value pending packet in the buffer is 2-competitive [11], [12]. For a variant in which the deadlines of the packets (weakly) increase with their release dates, Li et al. [16] propose an optimal deterministic ϕ-competitive algorithm. Using the same analysis, but in a more complicated way, Li et al. provide a $(3/\phi \approx 1.854)$-competitive deterministic algorithm [16] for the general model. Independently, Englert and Westermann present a 1.894-competitive deterministic memoryless algorithm and a $(2\sqrt{2} - 1 \approx 1.828)$-competitive deterministic algorithm [8]. Closing the gap $[1.618,\ 1.828]$ of competitive ratio for deterministic algorithms is a difficult open problem. A randomized online algorithm with a competitive ratio of $e/(e-1) \approx 1.582$ is proposed in [5]. The lower bound of competitive ratio of randomized algorithms is 1.25. How to tighten the gap $[1.25,\ 1.582]$ in the randomized bounded-delay model remains open. If the buffer size B is a finite number, the generalization of the bounded-delay model is called a *bounded-buffer model*. In [13], a 3-competitive deterministic algorithm and a $(\phi^2 \approx 2.618)$-competitive randomized algorithms are given. Fung [10] provides a 2-competitive deterministic algorithm and Li presents an alternative proof [15]. When the number of size-bounded buffers is more than 1, Azar and Levy [3] provide a 9.82-competitive deterministic algorithm and Li [14] improves the competitive ratio to $3 + \sqrt{3} \approx 4.723$.

B. Paper organization

In this paper, we first present the model that we study and the metric that we use to measure online learning algorithms in Section II. Then we introduce our online learning algorithms in Section III. The theoretical analysis of these online learning algorithms are presented in Section IV. Simulation results are provided in Section V. Finally, we conclude the paper in Section VI.

II. The Model

We study the following model for QoS buffer management. Time is discrete. A time step t represents the time interval $(t-1,\ t]$. Packets arrive over time. Each packet p has a non-negative value $v_p \in \mathbb{R}^+$ and an integer deadline $d_p \in \mathbb{Z}^+$. The deadline d_p specifies the time by which p should be sent. There is a buffer. Packets already existing in the buffers can be dropped at any time before they are served. A dropped packet cannot be delivered any more. In each time step, at most one packet from the buffer can be sent; the success of delivering a packet depends on the channel's reliability which is not known ahead. The objective is to maximize *weighted throughput*, defined as the total value of the transmitted packets by their respective deadlines. This model is called a *generalized model* and called *bounded-delay model* if the channel's reliability is always 1. Essentially, this buffer management is an online decision making problem.

In general, people consider the worst-case guarantees for online algorithms. Competitive analysis is one of the widely accepted way in measuring an online algorithm's worst-case performance in theoretical computer science and operations research. However, competitive analysis is too pessimistic as the adversary is empowered to change the input sequences according to what the online algorithm does over time. Many alternative metrics have been proposed (see [4] and the references therein). In this paper, we consider *external regret* as the metric in measuring online (learning) algorithms' performance.

Let $D = \{D_1,\ D_2,\ \ldots,\ D_n\}$ be the set of possible decisions in each time period. Denote the profit/gain incurred at time t from taking decision D_j by G_t^j. We assume throughout that profits are bounded; for example, $n \cdot v_{\max}$, where $v_{\max}$ is the maximal value of a packet.

Definition 1: **Regret** [9]. Any scheme (deterministic or randomized) for selecting decisions can be described in terms of the probability, μ_t^j, of choosing decision D_j at time t. Consider now a scheme S for selecting decisions. Let $\{\mu_t\}_{t\geq 0}$ be the probability weights implied by the scheme. Then, the expected gain from using S, $G(S)$, over T periods will be $\sum_{t=1}^{T} \sum_{D_j \in D} \mu_t^j G_t^j$. We define the *regret* incurred by S from using decision D_j to be $R_T^j(S) =$

$\max\{0,\ \sum_{t=1}^{T} \mu_t^j (G_t^j - G_t^i)\}$. The *regret* from using S will be $R_T(S) = \sum_{j \in D} R_T^j(S)$. The scheme S will have the *no internal regret* property if its expected regret is small, i.e., $R_T(S) = o(T)$.

III. Online Learning Algorithms

In Section III-A, we design an optimal offline algorithm for the generalized bounded-delay model. In Section III-B, we show the limit of online learning for the bounded-delay model, especially for the model in which the adversarial information is not revealed over time. This result can be extended to the generalized model. At last, we present a few online learning algorithms for the bounded-delay model and a few of its variants in Section III-C.

A. An optimal offline algorithm

We consider the offline setting of the generalized bounded-delay model, in which the success ratio of delivering a packet in a time step varies in the range of $[0,\ 1]$. We call this ratio *channel reliability*. This model is motivated by applications of wireless communications. The problem is stated as follows: Given a set of packets with all their characteristic known in advance, and a sequence of static channel states with known reliability, we are looking for an algorithm with the maximum total weight.

Theorem 1: The generalized bounded-delay model in which the channel reliability varies can be solved in pseudo-polynomial time.

Proof: If the buffer size B is infinite, an optimal offline algorithm has been given for the bounded-delay model in [12] with a running time of $O(n \log n)$. If the buffer size B is a constant, an optimal offline algorithm is proposed in [18] with a running time of $O(n^2 \log \max\{B,\ n\})$.

Now, we consider the generalization in which the channel reliability may not always be constant. The buffer size B is assumed infinite. Given a set of packets $P = \{p_1,\ \ldots,\ p_n\}$, each packet p_i is denoted as a 3-tuple $(r_i,\ d_i,\ v_i)$. Let $r_{\min} = \min\{r_i \mid i = 1,\ \ldots,\ n\}$ and $d_{\max} = \max\{d_i \mid i = 1,\ \ldots,\ n\}$. All possible scheduling time slots are in the range of $[r_{\min},\ d_{\max}]$; they are known as *feasible intervals*. Consider these m $(:= d_{\max} - r_{\min} + 1)$ time slots. Let the sequence of channel reliability be $\{\eta_1,\ \ldots,\ \eta_m\}$.

The reduction from the generalized bounded-delay model to the maximum weighted bipartite matching problem works as below. We first construct a graph $G = (V,\ E)$, and $V = L \cup R$ where $L \cap R = \emptyset$. Each $a_i \in L$ represents a packet p_i and each $b_j \in R$ represents a time slots in the feasible interval. Obviously, we have $|L| = n$ and $|R| = m$. Each vertex $a_i \in L$ represents the packet p_i. We create an edge connecting each a_i and each b_j with $r_{p_i} \le b_j \le d_{p_i}$. Also, each edge has an expected cost defined as $c(e_{ij}) = \eta_j \cdot v_{p_i}$, where η_j is the channel reliability at time slot represented by vertex b_j and v_{p_i} is the weight of the packet represented by vertex a_i. At this point, our problem could be solved by finding the maximum weighted matching of graph G and this results in a maximum expected total weight $\mathbf{E}(W)$.

The running complexity of this algorithm includes two parts: the construction part and the part of solving the reduced problem. The construction takes a linear time $O(|V| + |E|)$. Calculating a maximum weighted matching takes time $O(|E|^3)$, using the Hungarian algorithm. ■

B. The power of learning

After introducing the offline solution to the generalized bounded-delay model, we consider online solutions. We note that competitive online algorithms and online learning algorithms can be both applied as online approaches to this model without making any assumptions on packets' stochastic properties. The main question that should be answered is what is the power/limit that an adversarial input sequence may have. That is, does there exists a finite input instance such that any online learning algorithm cannot achieve a constant external regret?

Theorem 2: Consider a finite but large input sequence. There is no online learning algorithm that can have a constant external regret.

Proof: In order to show that for a given finite input sequence, no online learning algorithm has a constant external regret, we need to show that compared with the best algorithm in hindsight, any online learning algorithm has a constant > 1 competitive ratio. We consider a specific case in which all packets' deadlines are weakly increasing along their release time. We call such instances *agreeable deadline*. This example is modified from the one given in [6]. We first show that for any online policy π, its competitive ratio $c \ge \phi - \epsilon$, for any small $\epsilon > 0$.

At each time step t, two packets p_t and p'_t with *span* (the difference between deadline and release time) 1 and 2 respectively are released with values v_t and v_{t+1} respectively. Assume a deterministic online algorithm runs policy π, and its competitive ratio is c. Initially, we set $\tau = \sqrt{5} - 2$, $v_0 = 1$, $v_1 = \phi + \epsilon$, and $v_{i+1} = (v_i - v_{i-1})/\tau$. (Explicitly, we have $v_i = (1 - \epsilon)\phi^i + \epsilon(\phi + 1)^i$, $\forall i \ge 0$.)

1. Let k be a sufficient large number. If there exists $0 \le j < k$, π selects p'_j in step j (i.e., time interval $[j,\ j+1]$), its adversary stops releasing packets after j.
   π does not select p_j to send in step j, and $\forall i < j$, π selects p_i to send in step i. On the contrary, π's adversary delivers p'_i in step i, $\forall i < j$, p_j in step j, and p'_j in step $j+1$. Note that $\lim_{k\to\infty} \frac{v_k}{v_{k-1}} = \phi + 1$ and $\sum_{i=1}^{n} v_i = (v_{n-1} - v_0)/\tau$. The competitive ratio is:
   $$\begin{aligned} c &\ge \frac{(v_1 + v_2 + \cdots + v_{j+1}) + v_j}{(v_0 + v_1 + \cdots + v_{j+1}) - v_j} \\ &> 1 + \frac{2\tau}{1-\tau} - (\frac{2\tau}{1-\tau})^2 \epsilon > \phi - \epsilon. \end{aligned}$$
2. Otherwise, the adversary releases all packets p_i and p'_i up to step $k-1$ and at time k, only p_k is released.
   $\forall i < k$, π selects p_i to send in step i. π's adversary delivers all packets p'_i in step i up to step k, where p_k is sent. Assume k is large.

$$
\begin{aligned}
c \quad &\geq \quad \frac{v_1 + \cdots + v_k + v_k}{v_0 + v_1 + \cdots + v_k} \\
&= \quad 1 + \frac{v_k - v_0}{v_0 + v_1 + \cdots + v_k} \\
&\xrightarrow{k\to\infty} \quad 1 + \frac{(1+\phi)v_{k-1} - 1}{1 + \phi + \frac{v_{k-1}-1}{\tau}} \\
&\xrightarrow{k\to\infty} \quad 1 + \frac{\phi + 1}{\tau^{-1}} = \phi.
\end{aligned}
$$

Thus, no algorithm can achieve a competitive ratio better than $\phi > 1$. In turn, no online learning algorithm has a constant expected regret during the course of the whole schedule. The limit of online learning algorithm is due to the fact that the partial input sequence cannot reveal more information on the optimality of each static algorithm until the end of this input sequence. ∎

C. Online learning algorithms

We design online learning algorithms for both the bounded-delay model and its generalization, as well as a few of its variants. We name a static algorithm an *expert*.

Based on Occam's razor (a principle stating that "the simplest explanation is usually the correct one"), we interpret that in general, we prefer simpler explanations. Consider the model that we study. Packet values, packet deadlines, and channel reliability are the three factors that we need to consider in the online decision making procedure. Thus, the static algorithms should be simple functions on packet values, packet deadlines, and channel reliability only.

Let $v_{\min}$ and $v_{\max}$ denote the minimum value and the maximum value of a packet in the input sequence. First, we design a few experts (static algorithm with hindsight). Then we design online learning algorithms based on the observed performance of these experts. Note that the number of experts cannot be large since this value determines the running time of the online learning algorithm in each time step. Thus, we apply the well-known *geometric rounding technique*. Consider all distinct packet values $v_{\min} = v_1 < v_2 < \ldots < v_n = v_{\max}$. We let $v_{\max} = (1+\delta)^k v_{\min}$, where $k \leq n$. Then for each value v_j, it falls in the range of $[v_{\min}(1+\delta)^{i-1},\ v_{\min}(1+\delta)^i)$. Thus, we have $\log_{1+\delta} \frac{v_{\max}}{v_{\min}}$ distinct intervals. Let $M = \lceil \log_{1+\delta} \frac{v_{\max}}{v_{\min}} \rceil$. We first introduce the ways that how these M experts work. Then we represent our online learning algorithm under various scenarios respectively.

We have two phases of delivering a packet for each expert. In the first phase, the reliability of the channel is predicted. In the second phase, the expert chooses one pending packet to send. In selecting a packet to send, packet values play the role and we have two (families of) strategies.

1) A strategy based on absolute values: Sort all pending packets in increasing order of deadlines, with ties broken in favor of the one with a larger value. Choose the packet with a value $\geq$ its predefined threshold value. If such a packet is not available in its buffer, then this expert sends nothing. This algorithm is described in Algorithm 1.

Algorithm 1 EBAV($v_{\min},\ v_{\max},\ \delta,\ j \in [1,\ M]$)

1: Sort packets in the buffer in canonical order; i.e., in increasing order of deadlines with ties broken in favor of larger values.
2: Send the first packet p satisfying $v_p \geq v_{\max}/(1+\delta)^j$. If p does not exist in the buffer, then send nothing.

Each expert only admits the packet that it would send eventually. Thus, for the expert EBAV with a parameter j, it only accepts packets with values $\geq v_{\max}/(1+\delta)^j$. If the buffer size is limited, when overflow happens, we apply the greedy approach to filter out the packets that cannot be accommodated. That is, we drop the minimum-value packets when packet overflow happens; ties broken arbitrarily.

2) A strategy based on relative values: Sort all pending packets in increasing order of deadlines, with ties broken in favor of the one with a larger value. Choose the packet with a value $\geq$ its predefined threshold ratio than the maximum-value pending packet in the buffer, ties broken in favor of the earlier released packet. We note that such a packet is always in the buffer.

Algorithm 2 EBRV($v_{\min},\ v_{\max},\ \delta,\ j \in [1,\ M]$)

1: Sort packets in the buffer in canonical order; i.e., in increasing order of deadlines with ties broken in favor of larger values.
2: Let h be the maximum-value packet in the buffer.
3: Send the first packet p satisfying $v_p \geq v_h/(1+\delta)^j$. The packet p can always be found since either the first packet (in canonical order) or the packet h is the candidate.

In admitting packets, we apply the approach of identifying the *optimal provisional schedule*, which is similar to the one defined and used in competitive online algorithms. Given a set of pending packets P, a *provisional schedule* S specifies which packets in P should be sent in which time step. An *optimal provisional schedule* S^* is one that achieves the maximum weighted throughput among all provisional schedules on pending packets P (channel reliability is assumed 1). Clearly, an *optimal provisional schedule* S^* at time t can be calculated via a maximum-weighted bipartite matching over pending packets in $O(|P|^2)$ (see [13]).

3) Assume the channel reliability is always perfect: This is the typical bounded-delay model. Here we introduce two online learning algorithms based on the same strategy: follow the 'best expert'.

One is simply following the strategy of the expert who has the best gain up to the current time. The other one is following the strategy of the expert who has the best gain after delivering all the pending packets in its buffer successfully. We call these two online learning algorithms 'Follow COPT' and 'Follow OPT', respectively.

4) Assume the channel reliability varies over time: When the channel reliability is not a fixed value along all the time,

we need to consider the channel quality's variability as well. Thus, we employ a set of experts in predicting the channel's reliability over time.

In estimating the channel states, we introduce two experts. (We can adapt our algorithm to multiple experts.) Each expert insists on giving a fixed prediction of the future channel states, which is either H (representing 'high') or L (representing 'low'). These two experts are named EH and EL respectively. We then use the "weighted majority algorithm" [20], [21] in predicting the status of the channel reliability. We associate credits to these experts, which means to how much extend an expert's opinion could be trusted. Let the credit for expert EH at time t be c_t^{EH} and for expert EL be c_t^{EL} respectively. Initially, we set $c_0^{\mathrm{EH}} = c_0^{\mathrm{EL}} = 1$. In each time step $1,\ 2,\ \ldots,\ t-1$, we have a label indicting whether the channel reliability is 'H' or 'L'. Then we proceed as the following Winnow algorithm.

Algorithm 3 Winnow(EH, EL)

1: Initially, we set $c_0^{\mathrm{EH}} = c_0^{\mathrm{EL}} = 1$.
2: **if** $\sum c^{\mathrm{EH}} \geq \sum c^{\mathrm{EL}}$ **then**
3: Predict 'H'.
4: **else**
5: Predict 'L'.
6: **end if**
7: **if** the channel's quality was 'high' **then**
8: $c^{\mathrm{EH}} = 2c^{\mathrm{EH}}$;
9: $c^{\mathrm{EL}} = c^{\mathrm{EL}}/2$;
10: **else**
11: $c^{\mathrm{EH}} = c^{\mathrm{EH}}/2$;
12: $c^{\mathrm{EL}} = 2c^{\mathrm{EL}}$.
13: **end if**

However, in admitting packets, we are not only predicting the next step's channel reliability, we also need to estimate the future channel's status when we calculate an optimal provisional schedule. Hence, we need to study the 'chain effect' of the prediction. Let $E(\cdot)$ be the predicted value using the Winnow algorithm.

Lemma 1: If $E(c_t^{\mathrm{EH}}) \geq E(c_t^{\mathrm{EL}})$ (respectively, $E(c_t^{\mathrm{EH}}) < E(c_t^{\mathrm{EL}})$) holds at time slot t, then $E(c_{t+1}^{\mathrm{EH}}) \geq E(c_{t+1}^{\mathrm{EL}})$ (respectively, $E(c_{t+1}^{\mathrm{EH}}) < E(c_{t+1}^{\mathrm{EL}})$) holds at time $t+1$.

Proof: Let η_t denote the predicted reliability at time t. We have

$$\eta_t = \frac{E(c_t^{\mathrm{EH}})}{E(c_t^{\mathrm{EH}}) + E(c_t^{\mathrm{EL}})}.$$

Based on the Winnow Algorithm, we have

$$E(c_t^{\mathrm{EH}}) = \eta_{t-1}E(c_{t-1}^{\mathrm{EH}}) + (1-\eta_{t-1})\frac{E(c_{t-1}^{\mathrm{EH}})}{2}. \quad (1)$$

$$E(c_t^{\mathrm{EL}}) = \eta_{t-1}\frac{E(c_{t-1}^{\mathrm{EL}})}{2} + (1-\eta_{t-1})E(c_{t-1}^{\mathrm{EH}}). \quad (2)$$

Given the assumption that $E(c_t^{\mathrm{EH}}) \geq E(c_t^{\mathrm{EL}})$, we have $\eta_t \geq 1/2$. Then from Equations 1 and 2, we have

$$\begin{aligned} E(c_{t+1}^{\mathrm{EH}}) &= (1+\eta_t)\frac{1}{2}E(c_t^{\mathrm{EH}}) \\ &\geq \frac{3}{4}E(c_t^{\mathrm{EH}}). \\ E(c_{t+1}^{\mathrm{EL}}) &= \left(1-\frac{1}{2}\eta_t\right)E(c_t^{\mathrm{EL}}) \\ &\leq \frac{3}{4}E(c_t^{\mathrm{EL}}). \end{aligned}$$

Lemma 1 is completed. ■

Theorem 3 (Monotonicity of η): If the initial expected credits of experts satisfies $E(c_0^{\mathrm{EH}}) \geq E(c_0^{\mathrm{EL}})$ (respectively, $E(c_0^{\mathrm{EH}}) < E(c_0^{\mathrm{EL}})$), the reliability of channel η_t is non-decreasing (respectively, non-increasing) for t.

Proof: Theorem 3 can be proved using the inductive method. Based on the definition of η and $E(\cdot)$, we have

$$\begin{aligned} \eta_t &= \frac{E(c_t^{\mathrm{EH}})}{E(c_t^{\mathrm{EH}}) + E(c_t^{\mathrm{EL}})}. \\ \eta_{t+1} &= \frac{E(c_{t+1}^{\mathrm{EH}})}{E(c_{t+1}^{\mathrm{EH}}) + E(c_{t+1}^{\mathrm{EL}})} \\ &= \frac{(1+\eta_t)\frac{1}{2}E(c_t^{\mathrm{EH}})}{(1+\eta_t)\frac{1}{2}E(c_t^{\mathrm{EH}}) + \left(1-\frac{1}{2}\eta_t\right)E(c_t^{\mathrm{EL}})} \\ &= \frac{(1+\eta_t)E(c_t^{\mathrm{EH}})}{(1+\eta_t)E(c_t^{\mathrm{EH}}) + (2-\eta_t)E(c_t^{\mathrm{EL}})}. \end{aligned}$$

Therefore, we have

$$\begin{aligned} \frac{\eta_{t+1}}{\eta_t} &= \frac{(1+\eta_t)E(c_t^{\mathrm{EH}})}{(1+\eta_t)E(c_t^{\mathrm{EH}}) + (2-\eta_t)E(c_t^{\mathrm{EL}})}\left(\frac{E(c_t^{\mathrm{EH}}) + E(c_t^{\mathrm{EL}})}{E(c_t^{\mathrm{EH}})}\right) \\ &= \frac{(1+\eta_t)E(c_t^{\mathrm{EH}}) + (1+\eta_t)E(c_t^{\mathrm{EL}})}{(1+\eta_t)E(c_t^{\mathrm{EH}}) + (2-\eta_t)E(c_t^{\mathrm{EL}})}. \end{aligned}$$

Assume that we have the initial case of $E(c_0^{\mathrm{EH}}) \geq E(c_0^{\mathrm{EL}})$, then based on Lemma 1, we have $E(c_t^{\mathrm{EH}}) \geq E(c_t^{\mathrm{EL}})$. That is $1/2 \leq \eta_t \leq 1$. So $2-\eta_t \leq 3/4 \leq 1+\eta_t$, which indicates $(1+\eta_t)E(c_t^{\mathrm{EL}}) \geq (2-\eta_t)E(c_t^{\mathrm{EL}})$ ($E(c_t^{\mathrm{EL}}) > 0$). Thus, it is obvious that $\eta_{t+1}/\eta_t \geq 1$. Theorem 3 is completed. ■

Theorem 3 indicates that the predicted reliability of the channel at a certain time highly depend on the result predicted in the previous step. Thus, the 'predicted status' can be rolled over along the time we send packets. We have the following algorithm.

Algorithm 4 PCR(EH, EL)

1: Apply the Winnow Algorithm to predict channel's reliability.
2: **if** $E(c_t^{\mathrm{EH}}) \geq E(c_t^{\mathrm{EL}})$ **then**
3: Assign all predicted channel status 'High'.
4: **else**
5: Assign all predicted channel status 'Low'.
6: **end if**

After we identify the channel status over time, we can apply the bipartite matching to find the optimal provisional schedule and arrange packets in canonical order. In summary, we assign

all the pending packets to their latest time slots that they can be feasibly of being sent by deadlines if $E(c_t^{\text{EH}}) \geq E(c_t^{\text{EL}})$. If $E(c_t^{\text{EH}}) < E(c_t^{\text{EL}})$, these pending packets are assigned to the earliest time slots for delivery.

5) There are only two experts: When there are only two experts in predicting the channel's reliability, the online learning algorithm makes its decision based on these two experts and we consider this variant for analysis in Section IV.

D. An example

To better illustrate how our online learning approach work and its performance against the optimal offline algorithm and best known competitive online algorithms, we give the following example.

We assume the channel reliability is always 1. There are 4 packets p_1, p_2 , p_3 and p_4. Their release time, deadlines, and values are listed in Table I. Remember that packets are unit-length.

TABLE I
PACKETS' INFORMATION

packet	release time	deadline	value
p_1	0	1	2
p_2	0	2	5
p_3	2	3	4
p_4	2	4	7

We compare the following three kinds of algorithms based on this same input as below.

1) The optimal offline algorithm: It is obvious that the offline algorithm could send all the packets before their respective deadlines and obtain a maximum total weighted throughput. The packets are sent in the order of "$p_1,\ p_2,\ p_3,\ p_4$" and this yields total value of 18.

2) The competitive online algorithm: Here, we consider algorithms SEMI-GREEDY [17] and EDF$_\beta$ [17].

For this example, we let the algorithm SEMI-GREEDY work under a perfect channel state such that it sends the maximum-value packet in each time step. It sends p_2 at time 0 (p_1 expires) and p_4 at time 2 (p_3 expires). The total value SEMI-GREEDY gains is 12.

For the algorithm EDF$_\beta$ ($\beta = 2$), we send the earliest-deadline packet whose value if at least $1/\beta$ times of the maximum-value of a pending packet in the buffer. Packet p_2 will be sent at time 0 since $v_2 > \beta v_1 = 2v_1$. At time 2, EDF$_\beta$ chooses p_3 to send since $\beta v_3 = 2v_3 > v_4$. The packet p_4 will be sent afterwards. Therefore, EDF$_\beta$ ($\beta = 2$) gains a total value of 16.

3) The online learning algorithm: We use the algorithm EBAV as an example. The algorithm EBRV works in a similar way and we omit it here. We choose $\delta = 0.5$ as the input parameter of the algorithm. We also have $v_{\max} = 7$ and $v_{\min} = 2$. So the number of experts we will have is $M = \lceil \log_{1+\delta} \frac{v_{\max}}{v_{\min}} \rceil = 4$. Refer to Algorithm 1, we have the schedule list of each expert is shown in the table II.

The online learning algorithm EBAV runs the following strategy: Follow the expert who has the best gain up to the current time.

TABLE II
THE SCHEDULE LISTS OF EXPERTS

expert	$t=0$	$t=1$	$t=2$	$t=3$	value
expert 1	p_2	$-$	p_4	$-$	12
expert 2	p_2	$-$	p_3	p_4	16
expert 3	p_2	$-$	p_3	p_4	16
expert 4	p_1	p_2	p_3	p_4	18

Initially, the online learning algorithm simply uses EDF as the strategy since no expert exists to follow yet. Therefore, the online learning algorithm schedules p_1 at time $t = 0$. After all experts finish scheduling their first packets, expert 1, 2 and 3 are the best for step 1. Assume the online algorithm follows expert 1 at this step, then, it will schedule p_2 at $t = 1$ for that $v_2 > v_{\max}/(1+\delta)$. At the end of step 2, expert 4 has the best gain for the first time steps. The online learning algorithm switches to expert 4's strategy, which sends p_3 at time $t = 2$. The online algorithm follow the expert 4 for time $t = 2$ and $t = 3$. The total value that it gains is 18.

Table III summarizes the weighted throughput of these 4 different kinds of algorithms on the above example. The online learning algorithm achieves the optimal gain that can be achieved by an offline algorithm.

TABLE III
PERFORMANCE OF ALGORITHMS

Algorithm	Schedule	Value
offline	$p_1,\ p_2,\ p_3,\ p_4$	18
SEMI-GREEDY	$p_2,\ p_4$	12
EDF$_\beta(\beta = 2)$	$p_2,\ p_3,\ p_4$	16
EBAV	$p_1,\ p_2,\ p_3,\ p_4$	18

IV. ANALYSIS OF THE ONLINE LEARNING ALGORITHMS

We apply the local optimization technique to analyze the online learning algorithms that we develop. We first consider an instance with only two values for packets. Then we show that the regret of the online learning algorithm is bounded by a ratio of these two packets. Finally, we claim that this ratio can be generalized to the case in which packets can have multiple values.

Assume that there are only two kinds of packets denoted as p_s and p_b, where $v_s = 1$ and $v_b = \alpha > 1$. In this case, it is obvious that only two experts are needed. Expert 1 is denoted as EA and it will schedule the earliest deadline packet whose weight $\geq v_s$. If both p_s and p_b exists and if they share the same deadline, expert 1 will schedule p_b. Expert 2 is denoted as EB and it will schedule the earliest deadline packet whose weight $\geq \alpha$. If only p_s remains in the current buffer, expert 2 will schedule p_s.

Let the best expert among EA and EB be EXP. Let the total gains of the best expert and of the online learning algorithm are G^{EXP} and G^{ON} respectively. Then the average regret R per time step is expressed as

$$R = \frac{G^{\text{EXP}} - G^{\text{ON}}}{T}, \tag{3}$$

where T is the overall time spent in scheduling all the packets for the online learning algorithm.

We case study the regret R. Given a time step, if the online algorithm has exactly the same behavior (schedules the same packet) with the best expert in hindsight, then the average regret will be zero. Otherwise, we claim that there exist a time step t such that the packet scheduled by the online learning algorithm is different from the one scheduled by the best expert.

Assume the best expert EXP send a packet p_s and the online learning algorithm ON schedule a packet p_b and assume EXP outperforms ON.

Remark 1: The best expert must send a packet p_b in the future, i.e., after the time the online learning algorithm send the packet p_s.

Proof: We prove Remark 1 using the contradiction method. Assume that the claim fails at some time $t' > t$. Then, in any time step $t' > t$, the best expert sends a packet with a less or equal weight comparing to the online learning algorithm since no packet with value v_b is sent by EXP. Note that t is the first time step that the online learning algorithm differs from the best expert, the best expert will have a strictly less gain than the online learning algorithm. This contradicts the assumption that the best expert outperforms the online learning algorithm. ■

In this case, we charge EXP the value $1 + \alpha$ and charge ON the value α. We have an internal regret of value 1 and the ratio of gains is bounded by $(1 + \alpha)/\alpha$.

Assume the best expert EXP send a packet p_b and the online learning algorithm ON schedule a packet p_s.

In this case, the packet p_b is either sent by ON already or it is pending in ON's buffer.

If ON has sent this packet in a previous time step t', we can charge EXP value $\leq \alpha$ to t'. In the current step t, we charge EXP and ON values α and 1 respectively. Overall, the ratio of the charged values in t' and t is bounded by $\frac{\alpha+\alpha}{1+\alpha}$.

If ON schedules the packet p_b in a future time step t', this may incur a loss of value $\leq \alpha$ due to packet deadline constraint. We charge EXP and ON values α and α in t' respectively. Overall, the ratio of the charged values is bounded by $\frac{\alpha+\alpha}{1+\alpha}$.

This analysis approach can be applied to multiple values rather than two ones. We skip the details of expanding this analysis. We give a tight example for the online learning algorithm we present.

Example 1: We have two kinds of packets with values 1 and α respectively. Initially, B packets with value 1 arrive and their deadlines are $1,\ 2,\ \ldots,\ B$, and at the same time, B packets with value α arrive and their deadlines are $B + 1,\ B + 2,\ \ldots,\ 2B$. For the online learning algorithm EBRV (or EBAV), if it chooses the packets with value 1 to send (of course, it plans to send the packets with value α in the future), we generate an input of B packets with value α at time $B + 1$ and these packets have deadlines of $2B$. If the online learning algorithm chooses packets with value α to send in steps $1,\ 2,\ \ldots,\ B$, we stop releasing new packets. In either case, the average regret is $\frac{\alpha-1}{2}$ per time step.

V. Simulation

We develop a prototype to simulate the performance of online learning algorithms. There is a packet generator. This generator generates packets with release time, deadlines and values. In our setting, we assume all these numbers are generated uniformly randomly. In total, there are 2000 packets generated. The range of the release time is $[0,\ 999]$. The range of the deadlines is $[0,\ 999]$ and the range of the packet values is $[0,\ 99]$. The maximum and the minimum packet values are $v_{\max} = 99.951$ and $v_{\min} = 0.009$, respectively. For the current version of our simulation, the channel state is fixed. We design the offline algorithm as follows. We assume the channel reliability is always perfect. We implement EBAV and EBRV in Section III-C. In geometric rounding, we set $\delta = 0.5$. In total, there are 23 experts $(0.009(1 + 0.5)^{23} \approx 99.951)$.

In our simulated results, the best expert yields a total gain of 72096.615. It sends 1000 packets and it is the second expert. In each time step, the best expert chooses a packet with a value $\geq v_{\max}/[(1 + \delta)^2 v_{\min}]$ to send. The worst expert yields a total gain of 99.951 with only 1 packet scheduled — the most valuable packet. The expert setting the threshold as the minimum-value packet yields a total gain of 56711.178 and 1000 packets are scheduled successfully. The online learning algorithm who follows the best expert so far will gain a profit as 72050.509 with 999 packets scheduled. Thus, the regret of online learning algorithm is bounded by a value 52 ($<$ 0.00813) and it is almost the optimal expert.

Figure 2 shows the simulated results we have. In Figure 2, the dash-dotted line represents the performance of our online learning algorithm. In this figure, we take experts with index j of $0,\ 1,\ 2,\ 5,\ 20$ among all 23 experts.

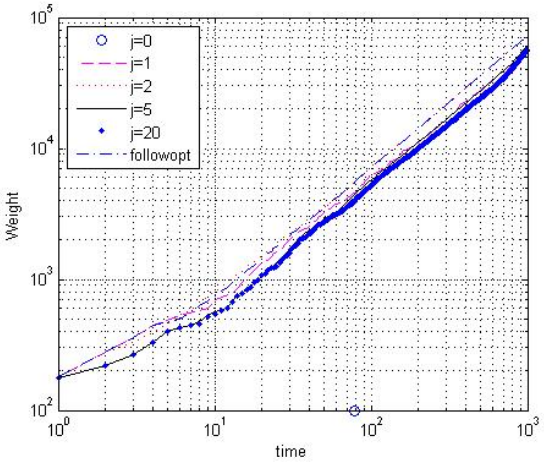

Fig. 2. Simulated results of the online learning algorithms

We also develop the prototype to simulate the performance of the optimal offline algorithm as well as two competitive

online algorithms: SEMI-GREEDY [17] and EDF$_\beta$ [17]. The simulation is run on the same data set used above. In our simulated results, the optimal offline algorithm yields a total gain of 73619.174. It sends 1000 packets. The algorithm SEMI-GREEDY yields a near optimal result 73330.578. EDF$_\beta$ earns a total gain of 72145.685, where $\beta = 2$. We notice that both online algorithms' performances are very close to the optimal offline solution.

Figure 3 shows the simulated results we have. In Figure 3, the dotted line represents the performance of our optimal offline algorithm, the dash-dotted line represents the performance of SEMI-GREEDY and the solid line represents the performance of EDF$_\beta$.

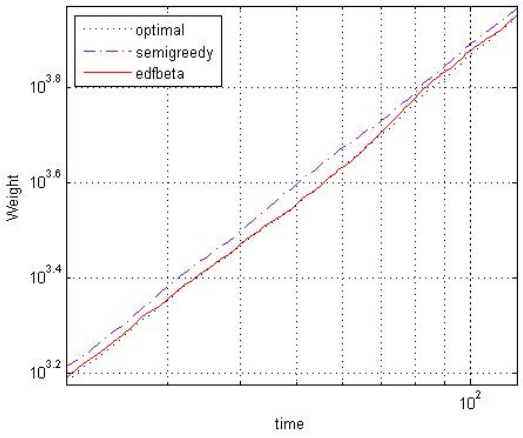

Fig. 3. Simulated results of the optimal offline algorithm and online algorithms

In summary, both online learning algorithms and competitive online algorithm perform nearly as good as the optimal offline algorithms. Our online learning algorithm has much less running complexity ($O(\log n)$ for each arriving packet) when compared with competitive online algorithms ($O(n)$ for each arriving packet).

VI. Conclusion

In this paper, we design algorithms to schedule packets with values and deadlines. This model has been studied extensively using competitive online algorithms. We apply the online learning approaches on this model as well as a few of its variants. These online learning algorithms' performance are analyzed theoretically in terms of external regret. We also measure these algorithms' performance experimentally. We conclude that no online learning algorithms have a constant regret. However, in general, our designed online learning algorithm works as almost well as the best known competitive online algorithm in practice. In our future work, we will study semi-online algorithms, assuming part of the input information is known.

Acknowledgement

The work is partially supported by NSF under grants CCF-0915681, CNS-0746649, and AFOSR under grant FA9550-09-1-0071.

References

[1] W. Aiello, Y. Mansour, S. Rajagopolan, and A. Rosen. Competitive queue policies for differentiated services. *in Proceedings of the 19th Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM)*, pages 431–440, 2000.

[2] N. Andelman, Y. Mansour, and A. Zhu. Competitive queuing polices for QoS switches. *in Proceedings of the 14th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 761–770, 2003.

[3] Y. Azar and N. Levy. Multiplexing packets with arbitrary deadlines in bounded buffers. *in Lecture Notes in Computer Science (SWAT)*, pages 5–16, 2006.

[4] A. Borodin and R. El-Yaniv. *Online Computation and Competitive Analysis*. Cambridge University Press, 1998.

[5] F. Y. L. Chin, M. Chrobak, S. P. Y. Fung, W. Jawor, J. Sgall, and T. Tichy. Online competitive algorithms for maximizing weighted throughput of unit jobs. *Journal of Discrete Algorithms*, 4:255–276, 2006.

[6] F. Y. L. Chin and S. P. Y. Fung. Online scheduling with partial job values: Does timesharing or randomization help? *Algorithmica*, 37(3):149–164, 2003.

[7] C. Dovrolis, D. Stiliadis, and P. Ramanathan. Proportional differentiated services: Delay differentiation and packet scheduling. *IEEE/ACM Transactions on Networking*, 10(1):12–26, 2002.

[8] M. Englert and M. Westermann. Considering suppressed packets improves buffer management in QoS switches. *in Proceedings of the 18th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 209–218, 2007.

[9] D. P. Foster and R. Vohra. Regret in the on-line decision problem. *Games and Economic Behavior*, 29:7–35, 1999.

[10] S. P. Y. Fung. Bounded delay packet scheduling in a bounded buffer. arXiv:0907.2741v1[cs.DS], July 2009.

[11] B. Hajek. On the competitiveness of online scheduling of unit-length packets with hard deadlines in slotted time. *in Proceedings of 2001 Conference on Information Sciences and Systems (CISS)*, pages 434–438, 2001.

[12] A. Kesselman, Z. Lotker, Y. Mansour, B. Patt-Shamir, B. Schieber, and M. Sviridenko. Buffer overflow management in QoS switches. *SIAM Journal on Computing (SICOMP)*, 33(3):563–583, 2004.

[13] F. Li. Competitive scheduling of packets with hard deadlines in a finite capacity queue. In *Proceedings of the 28th IEEE International Conference on Computer Communications (INFOCOM)*, pages 1062–1070, 2009.

[14] F. Li. Improved online algorithms for multiplexing weighted packets in bounded buffers. *Lecture Notes in Computer Science (AAIM)*, 5564:265–278, 2009.

[15] F. Li. Packet scheduling in a size-bounded buffer. arXiv:0909.3637[cs.DS], July 2009.

[16] F. Li, J. Sethuraman, and C. Stein. An optimal online algorithm for packet scheduling with agreeable deadlines. *in Proceedings of the 16th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 801–802, 2005.

[17] F. Li and Z. Zhang. Online maximizing weighted throughput in a fading channel. In *Proceedings of the 2009 IEEE International Symposium on Information Theory (ISIT)*, pages 591–595, 2009.

[18] F. Li and Z. Zhang. Scheduling weighted packets with deadlines over a fading channel. In *Proceedings of the 43rd Annual IEEE Conference on Information Sciences and Systems (CISS)*, pages 707–712, 2009.

[19] J. Liebeherr and N. Christin. A QoS architecture for quantitative service differentiation. *IEEE Communication Magazine*, 41(6):38–45, 2002.

[20] N. Littlestone. Learning quickly when irrelevant attributes abound: A new linear-threshold algorithm. *Machine Learning*, 2:285–318, 1988.

[21] N. Littlestone and M. Warmuth. The weighted majority algorithm. *Information and Computation*, 108(2):212–261, 1994.

[22] Y. Mansour, B. Patt-Shamir, and O. Lapid. Optimal smoothing schedules for real-time streams. *in Proceedings of the 19th Annual ACM Symposium on Principles of Distributed Computing (PODC)*, pages 21–29, 2000.

[23] V. Paxson and S. Floyd. Wide-area traffic: The failure of poisson modeling. *IEEE/ACM Transactions on Networking*, page 1995, 226-244.

[24] W. Willinger and V. Paxson. Where mathematics meets the Internet. *Notices of the American Mathematical Society*, pages 961–970, 1998.