Competitive Analysis of Buffer Policies with SLA Commitments

Boaz Patt-Shamir
School of Electrical Engineering
Tel Aviv University
Ramat Aviv 69978, Israel
boaz@eng.tau.ac.il

Gabriel Scalosub
Department of Computer Science
University of Toronto
Toronto, ON, Canada
scalosub@cs.toronto.edu

Yuval Shavitt
School of Electrical Engineering
Tel Aviv University
Ramat Aviv 69978, Israel
shavitt@eng.tau.ac.il

***Abstract*—We consider an abstraction of the problem of managing buffers where traffic is subject to service level agreements (SLA). In our abstraction of SLAs, some packets are marked as "committed" and the others are marked as "excess." The service provider must on one hand deliver all committed packets, and on the other hand can get extra revenue for any excess packet delivered. We study online algorithms managing a buffer with limited space, whose task is to decide which packets should be delivered and which should be dropped. Using competitive analysis, we show how to utilize additional buffer space and link bandwidth so that the number of excess packets delivered is comparable to the best possible by any off-line algorithm, while guaranteeing that no arriving committed packet is ever dropped. Simulations of such traffic (alone and combined with additional best-effort traffic) show that the performance of our algorithm is in fact much better than our analytical guarantees.**

I. Introduction

To support Quality of Service (QoS) in communication networks, service providers and customers typically have a Service Level Agreement (SLA) that specifies, on one hand, which minimal services the customer gets (e.g., what is the packet loss probability), and on the other hand, what is the maximal traffic pattern the customer injects (typically specified by a token-bucket descriptor). The provider reserves sufficient resources to support the promised QoS, and the customer should constrain its input to conform to the agreed characteristics. In many cases, the customer is allowed to inject "excess" traffic on top of the agreed ("committed") traffic. The provider may drop the excess traffic, or deliver some of it for an additional (per-megabyte) fee. Typically, multiple connections, some with two color marking, corresponding to the two types of traffic, and some with only best effort marking (corresponding to solely excess traffic), are handled together in one aggregate.

To facilitate the distinction between committed and excess traffic, many communication standards have provisions for a "rate meter" whose task is to mark packet conformance with the service level agreement (e.g., ATM [1], DiffServ [2], [3], [4], [5], MPLS [6], and—more recently—Metro Ethernet [7], [8]). In this paper we shall assume that the committed and excess packets of a flow are marked as "green" and "yellow," respectively.[1] Rate meters are deployed at the ingress of the network, where transmission rates are typically slower than the rates at the network core. Since color marking is in many cases infeasible at the high rates of core routers and switches, packets carry their color marks along, and routers differentiate the packet handling accordingly. Namely, green (committed) traffic should be assured a negligible drop probability, while yellow (excess) packets can be dropped freely.

As mentioned above, delivering excess traffic usually entails extra profit, and therefore, the service provider faces a dilemma: if yellow packets are delivered, additional profit can be made, but there is a risk of losing green packets, i.e., violating the SLA, potentially resulting in severe penalties. In this paper, we study algorithms to balance these two conflicting tendencies. Our approach is based on assuming that the resources at hand suffice to carry all committed traffic, and the goal is to maximize the amount of excess traffic delivered.

From the description above, it is clear that the provider would like to forward all of the green packet and as many yellow packets as possible. The simplest solution for this would be to assign a queue for each color type and to serve the queues with strict priority order. However, this solution is invalid since both types of packets might belong to the same flow and packets in a flow must be forwarded in order. Thus, we seek an algorithm that will allow us to accept yellow packets to the queue without dropping green packets, while ensuring that packets are forwarded in FIFO order.

A. Model

Concretely, our model is as follows (see Figure 1). The system we consider consists of a single queue denoted Q, and a fixed-rate outgoing link. Packets arrive at the queue arbitrarily. Each incoming packet is marked either as "green" (committed) or "yellow" (excess). We assume that all packets have equal size, and without loss of generality we let the size of packets be one unit. Packet arrival is adversarial, but we assume that the green packets can be served using a buffer of size B and link rate r. Namely, the committed traffic adheres

[1] In fact, the standards [3] distinguish between committed, excess, and violating packets, and mark them as green, yellow, and red, respectively. In this paper we assume that all violating (red) packets were already dropped, and focus only on committed (green) and excess (yellow) packets.

1-4244-2507-5/08/$20.00 ©2008 IEEE

to a token bucket of size B, corresponding to a maximum burst size, and rate r. This means that in every interval I of length T, the number of green packets arriving during I is at most $rT + B$. In particular, the above model captures scenarios where incoming traffic comprises of a single flow, or the more realistic case where the incoming traffic is an aggregate of multiple flows each with its own rate and burst parameters. In many real life scenarios some of the flows may have $r = B = 0$, namely they will consist of only best effort traffic.

The actual buffer size at the queue Q is $B_Q \geq B$, and the outgoing link rate is $r_Q \geq r$, so that if all yellow packets are dropped, the queue has sufficient resources in terms of buffer size and link rate such that no green packet should be dropped. (As we show later, even with a larger buffer size and a faster link, careless scheduling may result in loss of green packets due to overflow.)

We assume further that packets must be delivered in FIFO order, i.e., the order of packet delivery respects the order of their arrival (but some yellow packets may be missing from the output). Our algorithm has the nice feature of being work-conserving, i.e., the queue is never idle while its buffer is non-empty. Furthermore, our algorithm drops packets only from the tail of the queue, making it simple to implement.

An execution in our model proceeds in a sequence of discrete steps called *time slots*. Each time slot is divided into two substeps. The first substep is the *delivery substep*, where at most r_Q packets leave the queue from its head. In the second substep, called the *arrival substep*, traffic arrives at the system; then, at the discretion of the algorithm, some packets may be dropped, and the surviving new packets enter the queue at its tail. We note that the packets dropped may be from the set of newly arrived packets and packets already residing at the tail of the queue. In any case, the maximum number of packet that may be in the buffer after the arrival substep (i.e., between time slots) is at most B_Q. In particular, if the number of green packets residing in the queue after the delivery substep plus the number of green packets that arrive in the arrival substep is larger than B_Q, then some green packets will necessarily be lost due to overflow.

For simplicity we assume that arriving packets are handled in batches where all green packets are processed before all yellow packets. We further assume that in every time slot t, the algorithm may arrange the packets arriving at time t in any order. We note that our results, and specifically our algorithms, may be equivalently defined to deal with the case where this does not hold and packets are handled one by one.

We use competitive analysis in order to evaluate the performance of our algorithms [9], [10]. Formally, a schedule (i.e., a sequence of delivery times produced by an algorithm) is called *feasible* if it delivers all green packets. An algorithm is said to be *feasible* if it generates only feasible schedules. A feasible algorithm ALG is said to be *c-competitive* if for all traffic arrival sequences σ, the number of yellow packets delivered by ALG from σ is at least a c fraction of the best possible number of yellow packets delivered from σ under *any* feasible schedule (including schedules produced by off-line algorithms that know the future in advance). c is referred to as the *competitive ratio* or *competitive factor* of the algorithm. Obviously, $c \leq 1$ (1 means optimal). Our goal is to find a feasible algorithm whose competitive factor c is as large as possible.[2]

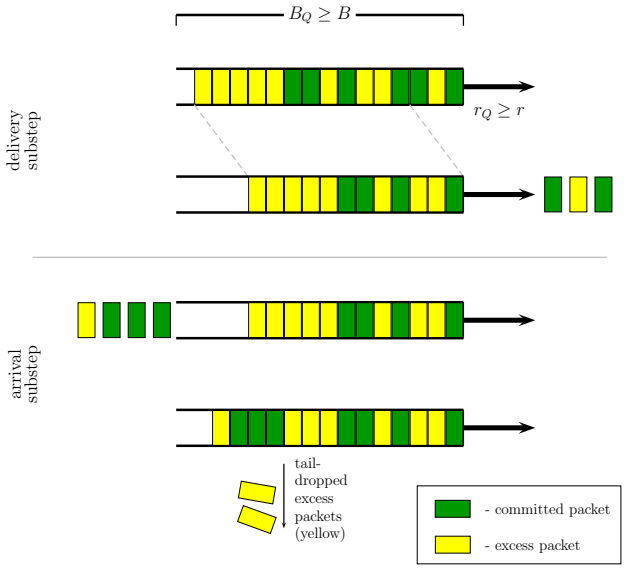

Fig. 1. Schematics of queue behavior in a single time slot: Queue Q uses a buffer of size $B_Q \geq B$ and has delivery rate $r_Q \geq r$ on its outgoing link. In the delivery substep of every time slot packets are delivered in the rate available (3 in the above example) in FIFO order, and remaining packets are advanced to the head of the queue. In the arrival substep packets arrive to the tail of the queue. Yellow packets residing in the tail of the queue might be dropped even if no overflow occurs. Newly arrived green packets are accepted before newly arrived yellow packets.

As customary in competitive analysis, we may view the on-line algorithm as competing against an off-line *adversary* that generates the input stream, and provides an optimal schedule for that input.

It is important at this point to distinguish between the competitive analysis used in this paper and other tools that are commonly used in this line of research, such as queueing theory [11] and fluid flow modeling [12]. While other tools study long term statistics and averages of the system performance, competitive analysis looks at the worst case analysis, thus achieving a good competitive ratio (like we do in this paper) enables us to guarantee good performance in *any* scenario, and saves us from low probability surprises.

B. Our Results

To start off our theoretical investigation, we show that without additional buffer space, no on-line algorithm that never loses a green packet has a competitive factor bounded away from zero. This result motivates our focus on the performance

[2]This definition usually corresponds to maximization problems, as is the case in the problem we consider. An analogous definition exists for minimization problems, where one aims to minimize the competitive factor.

of online algorithms that have more buffer space than the one available to the adversary. Our results can be viewed as an analysis of how to utilize the extra resources. Even when considering such cases where the algorithm has more resources than the ones available to the adversary, we show that for any $\varepsilon \in (0,1)$, no algorithm using space less than $(1+\varepsilon)B$ can have a competitive ratio better than ε (compared to the optimal performance possible using a buffer of size merely B). On the positive side, for any given $\varepsilon > 0$, we present an online algorithm which uses a buffer of size $(1+\varepsilon)B$, and outgoing link whose speed is s for some $s \geq r$. We prove that the number of yellow packets delivered by our algorithm for any packet arrival sequence is at least $\min\left\{\frac{\varepsilon}{1+\varepsilon-\frac{(s-r)}{B}}, 1\right\}$ times the maximum possible number of yellow packets delivered by any algorithm using a buffer of size B, and link rate r. These results are presented in Section II.

We further provide results of a simulation study, where we compare the performance of our algorithm with the popular "threshold" algorithm, which accepts yellow packets to the buffer only when the buffer occupancy is below some fixed threshold. Our simulations clearly show that our algorithm outperforms feasible threshold algorithms in a multitude of scenarios. Moreover, our results show that despite its "conservative" nature, our algorithm is robust, in the sense that it performs well even under high load, where the traffic contains intense excess traffic on top of the regulated committed traffic. Such scenarios occur when an aggregate contains many best effort flows. In all these cases, our algorithm throughput is close to the best throughput possible (that can be obtained only by a clairvoyant algorithm). These results are presented in Section III. We present some extensions and summarize in Section IV.

C. Previous Work

This paper extends the results of Cidon et al. [13] on protective buffer management. In [13], the two-color model is introduced, and a *protective* policy must deliver all green packets delivered by some reference process. Under this constraint, the goal is to maximize the number of other packets delivered.[3] The results in [13] include a characterization of protective policies, proofs that some natural policies are not protective, and a few algorithms that are protective, along with some numerical results comparing the performance of policies discussed.

Let us point out some of the main differences between our results and the ideas appearing in [13]. First, in this paper we use the tool of competitive analysis to compare algorithms, which allows us to quantify analytically the performance of algorithms. This goes beyond the qualitative distinction between protective and non-protective policies given by [13] (which correspond mostly to our notion of feasibility), or the numerical study they use to compare them. Another important feature of our paper is extending the model to the case of faster link: in [13], the reference process has smaller buffer space but the same link speed. Our paper shows how to utilize extra available bandwidth in addition to extra available space.

We note that the algorithm we propose in this paper enjoys the important feature of dropping packets only from the tail of the queue, while the algorithm in [13] requires a push-out buffer (i.e., dropping packets from anywhere in the buffer). Thus, our algorithm is much cheaper to implement in practice than the algorithm in [13], since it merely requires an additional pointer to the first (yellow) packet which might be dropped.

Other related work. Another interesting direction of research that was pursued more recently is assuming that each packet is assigned a real value, and the goal of the buffer management algorithm is to maximize the sum of values of delivered packets [14]. In the case of a single buffer, the best known competitive ratio for algorithms under general values is $\sqrt{3} \approx 1.732$, and the best lower bound is $1 + 1/\sqrt{2} \approx 1.707$ [15]. If packets may have only one of two values, 1 and $\alpha > 1$, then the competitive ratio is roughly 1.3 [15], and this is optimal [16], [17].

Packet marking has also been used as means of providing service differentiation to TCP flows [18], [19], by explicitly exploiting TCP characteristics. Active queue management (AQM) policies, such as random early drop (RED) have long been proposed to provide feedback to the senders, aiming at congestion avoidance in the network core [20]. Additional active queue management (AQM) policies have been applied to provide QoS guarantees such as delay and loss-ratio to excess real-time traffic (see e.g. [12]).

II. Analytical Study

A. Lower Bounds

We first consider the case where the queue has no extra buffer space or higher delivery rate, beyond the amount necessary to support the committed traffic, as defined by the rate and burst size to which the committed traffic adheres. We further assume w.l.o.g. that the rate in which committed traffic arrives is no greater than $r = 1$. The following theorem shows that additional buffer space is of the essence if an algorithm is to have a bounded competitive ratio.

Theorem 2.1: Let B denote the buffer space available to an optimal algorithm. Any online algorithm using a buffer of size $B' \leq B$ is either infeasible, or has competitive ratio arbitrarily close to zero.

Proof: First note that no algorithm using a buffer of size strictly less than B can be feasible. To see this note that any burst of B green packets arriving simultaneously cannot be accepted by any such algorithm, thus rendering it infeasible. We assume w.l.o.g. that the algorithm is work conserving. Assume the algorithm uses a buffer of size B, which is initially empty, and consider the arrival sequence consisting of one green packet p_0^g and one yellow packet p_0^y arriving at time 0,

[3] Superficially, the model in [13] may seem more foreign (e.g., green packets may be dropped if the reference process drops them), but essentially this is the same model we use. Cidon et al. used the green-red coloring which was common at the time, in this paper we chose to use the common terms of three color marking (green-yellow-red) which are commonly used now [3], [8]. Thus, our excess traffic is colored yellow and not red as in [13].

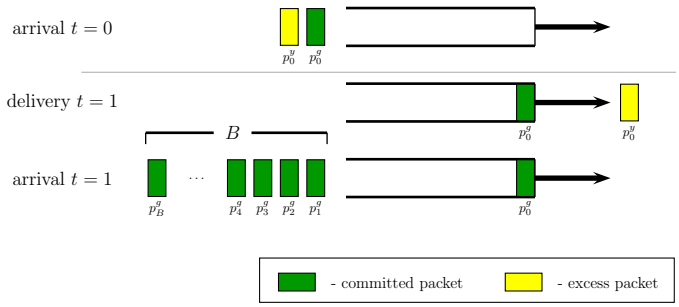

Fig. 2. The behavior of the adversary in case where the yellow packet is delivered at time $t = 1$, as depicted in the proof of Theorem 2.1.

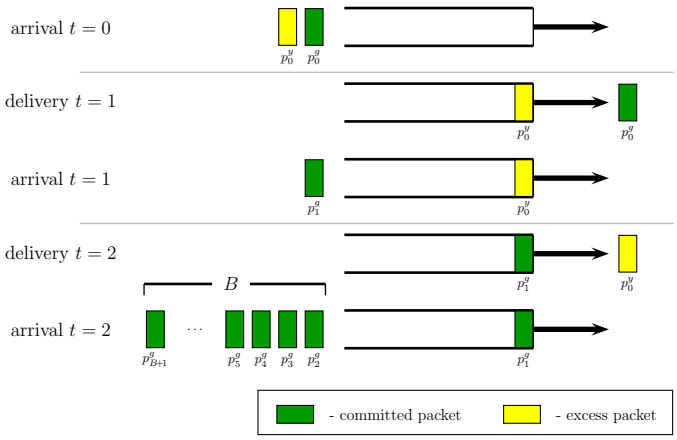

Fig. 3. The behavior of the adversary in case where the yellow packet is delivered at time $t = 2$, as depicted in the proof of Theorem 2.1.

and one green packet p_1^g arriving at time 1. If the algorithm forwards the yellow packet p_0^y, then by our above assumption it must do so at time $t \in \{1, 2\}$. We distinguish between two cases, according to the behavior of the online algorithm.

If the algorithm does not deliver the yellow packet by the end of time $t = 2$, then the sequence ends at time 2, and clearly the algorithm cannot be competitive, since an optimal algorithm would have forwarded all packets.

Assume on the other hand that the algorithm does deliver the yellow packet at time $t \in \{1, 2\}$. If $t = 1$, then consider the case where additional $B-1$ green packets $p_2^g, \ldots, p_B^g$ arrive at time 1. We therefore have green packet p_0^g still residing in the buffer of the algorithm at the end of the delivery substep of time 1, and an overall of B green packets – p_1^g and $p_2^g, \ldots, p_B^g$ – arriving at the arrival substep of time 1, which implies that the algorithm cannot store all these $B + 1$ green packets in its buffer, thus violating feasibility. Note that there exists a feasible schedule which can be obtained by rejecting the yellow packet upon arrival. This case is depicted in Figure 2.

Similarly, if $t = 2$, then the traffic continues by an arrival of B green packets at time 2. We therefore have green packet p_1^g still residing in the buffer of the algorithm at the end of the delivery substep of time 2, and an overall of B green packets arriving at the arrival substep of time 2, which again implies that the algorithm cannot be feasible. As before, rejecting the yellow packet could have resulted in a feasible schedule. This case is depicted in Figure 3. ■

Theorem 2.1 motivates considering the competitive ratio attainable by an online algorithm which is equipped with a larger buffer than the one available to the adversary. In what follows we assume the algorithm has a buffer of size $(1+\varepsilon)B$, for some $\varepsilon > 0$, and we compare its performance to any optimal algorithm using a buffer of size B. The following theorem show that even when equipped with additional buffer space, it might not be possible for an online algorithm to guarantee the optimal excess throughput possible with the minimum amount of buffer space necessary to support the committed traffic. As in the previous case, we assume w.l.o.g. that the rate in which committed traffic arrives is no greater than $r = 1$.

Theorem 2.2: For any $0 < \varepsilon < 1$, any feasible deterministic online algorithm using a buffer of size less than $(1 + \varepsilon)B$ cannot have a competitive ratio better than ε.

Proof: Consider any deterministic online algorithm which uses a buffer of size $(1 + \varepsilon)B$, and delivers all green packets. Assume the following arrival sequence to an empty buffer: At time 0 we have a burst of B yellow packets arriving, followed by one green packet arriving in every time step starting at time 1, until time εB. Consider time $t = \varepsilon B + 1$. We distinguish between two cases, according to the behavior of the online algorithm.

If by the end of time t the algorithm has already delivered the first green packet, then by the fact that the algorithm uses a FIFO discipline, it must have dropped at least $(1-\varepsilon)B$ of the yellow packets which arrived at time 0, and therefore cannot deliver more than εB yellow packets. An optimal schedule would have kept all yellow packets, and would have sent every green packet exactly B time steps after its arrival, while using a buffer size no greater than B. Such a policy would have delivered all B yellow packets. It follows that the ratio between the throughput of the algorithm and that of an optimal schedule is at most ε in this case.

Assume on the other hand that the algorithm has not yet delivered the first green packet by the end of time step t. Consider at this time t an arrival of a burst consisting of B green packets to the buffer. Since the algorithm still has all εB green packets injected prior to t in its buffer, and since it uses a buffer of size less than $(1 + \varepsilon)B$, it cannot store all newly arrived B green packets in its buffer, and therefore cannot be feasible. Note that feasibility could have been maintained by a work conserving FIFO policy which would have rejected all yellow packets upon arrival, while using a buffer of size B. It follows that any online algorithm that is to maintain feasibility must send the first green packet by time εB, and therefore the ratio between its performance and that of an optimal algorithm cannot be better than ε. ■

B. Upper Bounds

1) Algorithm Description: In this section we present an algorithm for our problem, and explore the effect of additional

buffer space and higher delivery rate on its performance. We assume discrete arrival times, and that in every time step t the adversary may deliver some $r \in \mathbb{N}$ packets while using a buffer of size B, whereas our algorithm can deliver some $s \geq r$ packets, $s \in \mathbb{N}$, while using a buffer of size $(1+\varepsilon)B$. As in the previous sections, here B and r are the maximal burst size and the maximum arrival rate of committed traffic.

Our algorithm is close in spirit to the Extended SPP algorithm appearing in [13]. We begin by defining a *simulator*, SIM, which is a buffer management algorithm that only performs admission control. SIM accepts only green packets, is work-conserving, and uses the minimum amount of buffer space and rate so as to be able to support the entire committed traffic. The simulator therefore works with a buffer of size B, and delivery rate r.

To allow us to discuss the execution of an algorithm, we define the following notation. Given an algorithm ALG and time t, $\mathcal{B}_{ALG}(t)$ denotes the state of the buffer (position of each packet) managed by ALG at the end of time t, i.e., after the arrival substep. We sometimes abuse notation slightly and use $\mathcal{B}_{ALG}(t)$ also to denote the *set* of packets residing in ALG's buffer at this time. Given any algorithm ALG, any time t such that $\mathcal{B}_{\mathrm{ALG}}(t) \neq \emptyset$ at the end of time t, and every green packet p that has arrived by t, we let $d_t^{\mathrm{ALG}}(p)$ be its *head-of-buffer distance*, i.e., the number of packets stored in the ALG's buffer before p, at time t (the exact meaning of whether this refers to the end of the delivery substep, or the end of the arrival substep, will be made clear from the context). If p has already been delivered by ALG, we let $d_t^{\mathrm{ALG}}(p) = 0$. For every such packet p, we define its *lag* at the end of time t by

$$\mathrm{lag}_t^{\mathrm{ALG}}(p) = \max\left\{d_t^{\mathrm{ALG}}(p) - d_t^{\mathrm{SIM}}(p), 0\right\},$$

i.e., how far ahead is a green packet in SIM's buffer compared to its position in ALG's buffer. Note that for any algorithm with a delivery rate of $s \geq r$, it might be the case that at some time t there are green packets yet undelivered by the simulator that have already been delivered by the algorithm. Furthermore, the fact that $s \geq r$ also implies that the lag of any green packet in the queue can never increase in subsequent time steps. Given $\varepsilon > 0$, we say an algorithm maintains the *ε-lag property*, if at any time t, and for any green packet p that has arrived by time t, $\mathrm{lag}_t^{\mathrm{ALG}}(p) \leq \varepsilon B$. At any time t, let A_t^G denote the set of green packets arriving at t, and let A_t^Y denote the set of yellow packets arriving at t. We further let $G_t^{ALG} \subseteq \mathcal{B}_{ALG}(t)$ denote the set of green packets residing in the buffer of ALG at the end of the delivery substep of time t. In what follows we sometimes omit the superscript/subscript ALG when the algorithm in question is clear from the context. Our algorithm is work conserving, and follows a FIFO discipline. It follows that we need only specify the behavior of our algorithm in the arrival substep of every time slot. Algorithm 1 gives the description of our online algorithm, ON.

The following lemma proves the correctness of ON by showing that it never uses a buffer of size greater than $(1+\varepsilon)B$,

Algorithm 1 $\mathrm{ON}(\varepsilon, s)$: at the end of the delivery substep of any time t,

1: Let A_t^G and A_t^Y be the set of green packets and set of yellow packets arriving at t, respectively.
2: Let G_t be the set of green packets in ON's buffer.
3: Let $m = \max\{\mathrm{lag}_t(p) \mid p \in G_t\}$ if $G_t \neq \emptyset$, and $m = 0$ otherwise.
4: **if** $A_t^G \neq \emptyset$ **then**
5: **if** the last packet in the buffer is yellow **then**
6: Let ℓ be the length of the maximal continuous block of yellow packets in the tail of the buffer.
7: Tail drop $d = \max\{\ell - (\varepsilon B - m), 0\}$ yellow packets.
8: **end if**
9: Accept A_t^G.
10: **end if**
11: Accept as many packets from A_t^Y as long as buffer occupancy is at most $(1+\varepsilon)B$.

and it accepts all green packets. We initially assume ON has an unbounded buffer. Furthermore, w.l.o.g., ON accepts the packets in A_t^G in the same order in which they are accepted by SIM.

Lemma 2.3 (Correctness): At any time t, ON maintains the ε-lag property, accepts all green packets, and never holds more than $(1+\varepsilon)B$ packets in its buffer.

Proof: Proof is by induction on t. For $t = 0$ the claim clearly holds since ON accepts A_0^G (since $\left|A_0^G\right| \leq B$ by the assumption that traffic is regulated by a token bucket with burst size B), and places any additional packets in A_0^Y at the tail of the buffer, i.e., after the packets in A_0^G. It follows that all green packets in A_0^G have zero lag at time t. By the definition of ON it accepts additional yellow packets to the extent that its occupancy does not exceed $(1+\varepsilon)B$.

Assume the claim holds for $t-1$, and consider time t. By the induction hypothesis, every green packet p in the buffer at time $t-1$ satisfies $\mathrm{lag}_{t-1}(p) \leq \varepsilon B$. This specifically holds for the last green packet p in ON's buffer at time $t-1$. If we drop at time t the entire block of yellow packets residing in the tail of the buffer at time $t-1$, then by the definition of lag, all packets in A_t^G will have lag at most $\mathrm{lag}_{t-1}(p)$, which by the induction hypothesis is at most εB. It follows that d is the minimal number of yellow packets residing in the tail of the buffer at time $t-1$, such that preempting this amount would ensure all packets in A_t^G still satisfy the ε-lag property after being admitted to the buffer. By the definition of ON, this is exactly the amount of yellow packets preempted at time t, hence, the algorithm maintains the ε-lag property at time t. Note that by the ε-lag property, any green packet in ON's buffer is at most εB packets behind its position in SIM. Since ON uses a buffer of size $(1+\varepsilon)B$ whereas SIM uses a buffer of size B, it follows that no green packet is ever positioned in a place greater than $(1+\varepsilon)B$ (since by the feasibility of SIM, it accepts all green packets). Since ON only places yellow

packets at the tail of the buffer as long as buffer occupancy does not exceed $(1+\varepsilon)B$, it follows that ON never uses a buffer of size greater than $(1+\varepsilon)B$, and is able to accept all green packets. ■

2) Performance Analysis of ON: In this section we analyze the performance of our proposed algorithm, and give explicit guarantees on its competitive ratio. Let ε be the percentage of additional buffer space available to ON, compared to the minimal buffer space necessary to deliver all committed traffic, and let s be its delivery rate, which is at least as high as the minimum rate r necessary to deliver all committed traffic. Let

$$c(\varepsilon, r, s) = \frac{\varepsilon}{1+\varepsilon-\frac{(s-r)}{B}}.$$

We prove the following theorem:

Theorem 2.4: Algorithm ON is $\min\{c(\varepsilon, r, s), 1\}$-competitive.

We first present some definitions which will be used throughout the performance analysis of ON.

Definition 2.5: Given any yellow packet p accepted by ON at time t, we say p is *safe* if there exists some minimal time $t' > t$ such that $A_{t'}^G \neq \emptyset$, and $p \in \mathcal{B}_{\mathrm{ON}}(t')$. In such a case we further say that p *turns safe* at time $t'+1$.

The above definition implies that a yellow packet residing in the buffer turns safe, the moment a later-arriving green packet is accepted to the buffer. Note that by specification, ON only drops yellow packets residing in its tail. It therefore follows that no safe packet is ever preempted by ON, which implies that all safe packets are eventually delivered by ON.

We denote by S_t the set of yellow packets which turn safe at time t. Given any time interval I, Let $S(I) = \cup_{t\in I} S_t$ denote the set of yellow packets turning safe at some point during I.

Similarly to the definition of lag in the previous section, we define the *half-step lag*, H-lag, of any green packet p in the buffer of ON after the delivery substep of time t by

$$\text{H-lag}_t(p) = \max\left\{d_t^{\mathrm{ALG}}(p) - d_t^{\mathrm{SIM}}(p), 0\right\}.$$

Note that for every time t and packet p for which the half-step lag is defined satisfies $\text{H-lag}_t(p) = \text{lag}_t(p)$. However, the half-step lag is only defined for green packets which are not sent at the time step after their arrival.

For every time t, we define

$$\phi(t) = \begin{cases} \max_{p\in G_t}\{\text{H-lag}_t(p)\} & G_t \neq \emptyset \\ 0 & \text{otherwise.} \end{cases}$$

Clearly for any time t, and any two green packets p, p' for which the half-step lag is defined, if p is ahead of p' in terms of buffer position, then $\text{H-lag}_t(p) \leq \text{H-lag}_t(p')$. It therefore follows that at any time t, the value of ϕ is determined by the half-step lag of the last green packet in ON's buffer at time t, if such a packet exists. We say ON is *reset at (the delivery substep of) time* t if $\phi(t) = 0$.

Let us first present a high-level description of the analysis; we consider overloaded time intervals during which yellow packets are dropped from the buffer, and during which no reset occurs. We show that during any such time interval, the number of safe packets in ON's buffer is sufficiently large, compared to the maximum number of yellow packets an optimal policy could have accepted. We show this by giving a lower bound on the committed traffic that must be handled by any algorithm during any such interval. This particularly implies that any optimal policy must dedicate sufficient resources (in terms of delivery rate and buffer size) in order to satisfy feasibility, thus leaving relatively few resources to deliver excess traffic. We further note that during the periods between overloaded time intervals, ON performs as well as any optimal policy.

As the starting point of our analysis, the following lemma gives some characterization of a reset event.

Lemma 2.6: If at any time t ON delivers less than s packets, then ON is reset at t.

Proof: Since ON is work conserving, and sends less than s packets in time t, it must follow that by the end of the delivery substep of time t, ON's buffer is empty. Therefore, by the definition of ϕ, we are guaranteed to have $\phi(t) = 0$, which implies that ON is reset at t. ■

We now turn to formally define overloaded time intervals. Consider any time t for which $\phi(t) = 0$. Starting from any such t, clearly as long as no yellow packets are preempted by ON then ON accepts at least as many yellow packets as the adversary does, since ON uses at least as much buffer space as the adversary does, and they both must accept all green packets. Consider any time t_0 in which a preemption occurs in ON. It follows that $\phi(t_0) = \varepsilon B$, since preemption only occurs in order to maintain the ε-lag property.

Let $t_1 < t_0$ be the latest time prior to t_0 where ON is reset. Note that such a time exists since ON is reset before the packets begin arriving. Furthermore, let $t_2 > t_0$ be the earliest time after t_0 where ON is reset. Note that such a time exists since if we denote by t' the time of the last arrival in the input sequence, then we are guaranteed to have a reset by time $t' + (1+\varepsilon)B$. Let $I_{t_0} = (t_1, t_2]$. We refer to any such interval as an *overloaded interval*. By contrast, we refer to any interval not contained in any overloaded interval as a *regular interval*. By the discussion presented above, during a regular interval ON accepts at least as many excess packets as the optimal schedule does. We therefore focus our attention on overloaded intervals. Note that the above definitions imply that for any two preemption events occurring at times t_0 and t_0', either $I_{t_0} = I_{t_0'}$, or $I_{t_0} \cap I_{t_0'} = \emptyset$. We will therefore refer to any such interval according to the first preemption event to which it corresponds, i.e., in an overloaded interval I_{t_0}, the first preemption event occurs at time t_0. Figure 4 gives an intuition as to the values of $\phi(t)$, and the decomposition of time into regular intervals, and overloaded intervals.

In order to simplify the remaining parts of our analysis, we assume that in every time slot t during an overloaded interval, either ON delivers exactly s packets in t, or it is idle in t. Our guarantees hold for the case where this does not hold, but the proof is much more involved, where every delivery substep should be divided into several slots, and each such slot must be evaluated independently. It should be noted that

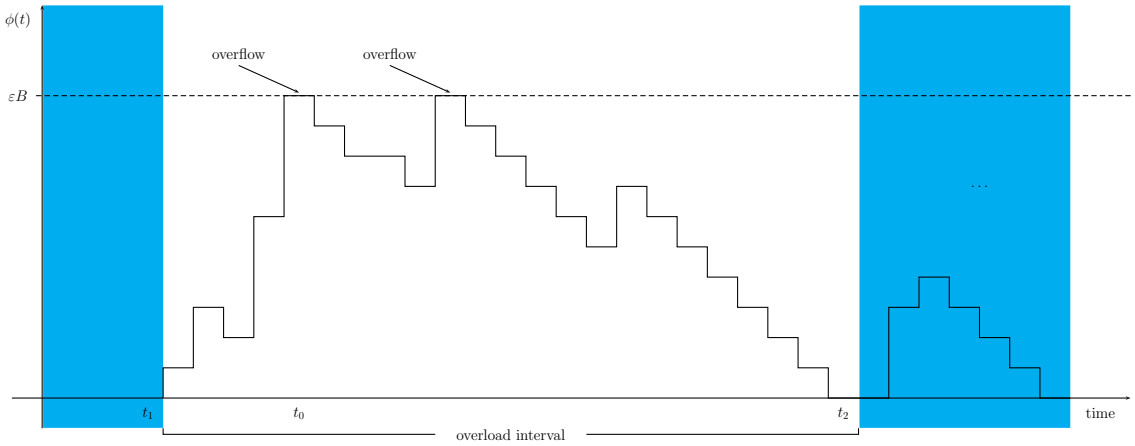

Fig. 4. Schematics of the value of $\phi(t)$. Note that there may be more than one overflow (i.e., a drop of yellow packets from the tail of the buffer) associated with a single overloaded interval. Furthermore, note that the marked interval on the right is not an overloaded interval, although ϕ does take strictly positive values in that interval. This is because there is no overflow associated with any time point in this interval.

by Lemma 2.6 the only difference involves the behavior of the algorithm at the very last time step of an overloaded interval, since at any other time step in an overloaded interval we are guaranteed that exactly s packets are delivered by ON. In any case, the key elements of the proof for the unrestricted case, and the case satisfying the above assumptions, are the same.

The following lemma characterizes the events which determine changes in the value of $\phi(t)$ during some overloaded interval I_{t_0}, and its dependance on the number of packets delivered by the simulator and the number of safe packets. In what follows, recall that S_t denotes the set of packets turning safe at time t.

Lemma 2.7: Let I_{t_0} be any overloaded interval. Given any time $t \in I_{t_0}$ for which $\phi(t) > 0$, if r_t is the number of packets sent by SIM at time t, then

$$\phi(t) = \phi(t-1) + |S_t| - (s - r_t).$$

Proof: Recall that we consider the case where ON delivers exactly s packets in every time slot during an overloaded interval.

If $A^G_{t-1} = \emptyset$, then $|S_t| = 0$, since yellow packets can turn safe only upon the arrival of green packets. Since $\phi(t) > 0$, then it must hold that the last green packet in $\mathcal{B}_{\mathrm{ON}}(t-1)$ is still in the buffer at the end of the delivery substep of time t, and it has advanced exactly s places in ON's buffer during this substep. On the other hand, this packet has advanced exactly r_t places in SIM's buffer.[4] Since ϕ is determined by the position of the last green packet in ON's buffer at the end of the delivery substep, it follows that

$$\phi(t) = \phi(t-1) + 0 - (s - r_t) = \phi(t-1) + |S_t| - (s - r_t).$$

[4]Note that the last green packet at a certain point is defined by the arrival sequence, and not by the algorithm managing the buffer. It follows that the last green packet in ON's buffer is also the last (possibly already sent) green packet in SIM's buffer. In case this packet has already been sent by SIM, then we simply have $r_t = 0$.

Assume that $A^G_{t-1} \neq \emptyset$ and consider the last green packet p in ON's buffer at the end of time step $t-1$. When considering ON's buffer at the end of time $t-1$, compared to the position of the last green packet after the delivery substep of $t-1$, the position of p has increased by exactly $|S_t|$.[5] Since $\phi(t) > 0$ we are guaranteed that p is still in ON's buffer after the delivery substep of time t. As in the previous case, during this substep, it has advanced s positions in ON's buffer, and r_t positions in SIM's buffer. Since ϕ is determined by the position of the last green packet in ON's buffer at the end of the delivery substep, it follows that

$$\phi(t) = \phi(t-1) + |S_t| - (s - r_t),$$

as required. ∎

Lemma 2.8: Let I_{t_0} be any overloaded interval, and let R_{t_0} denote the number of green packets delivered by SIM during I_{t_0}. Then

$$R_{t_0} = s\,|I_{t_0}| - S(I_{t_0}).$$

Proof: Given any time t during I_{t_0}, let $\Delta(t) = \phi(t) - \phi(t-1)$ denote the change in the value of ϕ between time $t-1$ and time t. By Lemma 2.7 we have

$$\Delta(t) = |S_t| - (s - r_t),$$

where r_t is the number of packets delivered by SIM at time t. Since by the definition of $I_{t_0} = (t_1, t_2]$, $\phi(t_1) = \phi(t_2) = 0$, we are guaranteed to have $\sum_{t \in I_{t_0}} \Delta(t) = 0$. Since

$$\begin{aligned}
\sum_{t \in I_{t_0}} \Delta(t) &= \sum_{t \in I_{t_0}} (|S_t| - (s - r_t)) \\
&= \sum_{t \in I_{t_0}} |S_t| - \sum_{t \in I_{t_0}} s - \sum_{t \in I_{t_0}} r_t \\
&= S(I_{t_0}) - s\,|I_{t_0}| + R_{t_0}
\end{aligned}$$

[5]Note that if there is no green packet in ON's buffer after the delivery substep of $t-1$, then ON is reset at $t-1$, in which case $\phi(t-1) = 0$.

we can conclude that $R_{t_0} = s\,|I_{t_0}| - S(I_{t_0})$. ∎

The following lemma concludes the proof of Theorem 2.4.

Lemma 2.9: Let I_{t_0} be any overloaded interval. Then, the ratio between the number of yellow packets admitted to the buffer by an optimal schedule during I_{t_0} and the number of packets turning safe during this interval, is at least $\min\{c(\varepsilon, r, s), 1\}$.

Proof: First note that by the fact that an optimal delivery strategy must correspond with token-bucket parameters of r (for the rate) and B (for the burst size), the overall number of packets, of any type, that can be accepted by an optimal solution during I_{t_0} is at most $r\,|I_{t_0}| + B$. Furthermore, by Lemma 2.8 we are guaranteed that SIM has delivered $s\,|I_{t_0}| - S(I_{t_0})$ green packet during I_{t_0}. By the fact that any optimal solution never delivers a green packet before its delivery time by SIM (since they both work at the same rate, and SIM never accepts yellow packets) we are guaranteed that at least this amount of green packets are handled by every optimal solution during I_{t_0}. It therefore follows that the overall number of yellow packets handled by any optimal solution during I_{t_0} is at most

$$\begin{aligned} r\,|I_{t_0}| + B - R_{t_0} &= r\,|I_{t_0}| + B - s\,|I_{t_0}| + |S(I_{t_0})| \\ &= |S(I_{t_0})| - (s-r)\,|I_{t_0}| + B. \quad (1) \end{aligned}$$

Since at any time t, $|S_t| \leq \varepsilon B$ (otherwise, we would have violated the ε-lag property), we are guaranteed to have $|I_{t_0}| \geq \frac{|S(I_{t_0})|}{\varepsilon B}$. By equation (1) we are guaranteed that the number of yellow packets handled by any optimal solution during I_{t_0} is at most

$$|S(I_{t_0})| - \frac{s-r}{\varepsilon B}\,|S(I_{t_0})| + B = \left(1 - \frac{s-r}{\varepsilon B}\right)|S(I_{t_0})| + B.$$

By the definition of I_{t_0}, we must have $|S(I_{t_0})| \geq \varepsilon B$. This follows from the fact that any unit increase in a packet's lag is only due to ON having turned safe an additional yellow packet.

The above implies that the ratio between the number of safe yellow packets accumulated during I_{t_0} and the number of yellow packets accepted by any optimal solution during I_{t_0}, is at least

$$\begin{aligned} \frac{|S(I_{t_0})|}{\left(1 - \frac{s-r}{\varepsilon B}\right)|S(I_{t_0})| + B} &\geq \frac{\varepsilon B}{\left(1 - \frac{s-r}{\varepsilon B}\right)\varepsilon B + B} \\ &= \frac{\varepsilon}{1 + \varepsilon - \frac{(s-r)}{B}} \end{aligned}$$

as required. ∎

Combining Lemma 2.9 with the fact that during regular intervals, ON does not drop any yellow packets, and therefore delivers at least as many yellow packets as any optimal solution, we obtain that the competitive ratio of ON is $\min\{c(\varepsilon, r, s), 1\}$.

3) Implementation Issues of ON: Our analysis implies that the maximum lag of a green packet in the system is exactly the lag of the last green packet stored in the buffer. Upon the arrival of a new green packet, its lag can be calculated in $O(1)$ time from the lag of the previous green packet and the number of yellow packets stored between them. This makes the calculation of m in line 3 trivial.

To be able to implement the algorithm we need to maintain a pointer to the first yellow packet that will cause the violation of the ε-lag property if a green packet arrives.

III. Simulation Study

In this section, we examine the effect of extra buffer space and higher delivery rates on the ability to obtain a higher throughput of excess traffic. We consider two algorithms, and conduct extensive simulations in order to evaluate their performance. Following the notation of the previous sections, we consider traffic consisting of both committed and excess packets, where the committed traffic conforms with token-bucket parameters of r (rate) and B (burst size). The first algorithm we consider is algorithm ON described in Section II. The second algorithm we consider is the Threshold algorithm which accepts yellow packets into the queue as long as buffer occupancy is below some threshold level T (see, e.g., [13]). We provide both algorithms with additional buffer space (corresponding to the quantity εB that was used in the previous sections), and higher delivery rates (corresponding to the quantity $s \geq r$ that was used in the previous sections).

For both algorithms we consider the ratio between their throughput and the one obtained by an upper bound on the best obtainable throughput achieved by any (possibly clairvoyant) algorithm. This value can be considered as the normalized throughput of the algorithms. We note that this upper bound might not be tight. The upper bound is obtained by an algorithm that is allowed to maintain two separate queues, one for the green traffic and one for the yellow traffic such that the overall occupancy in both queues does not exceed $(1+\varepsilon)B$. The algorithm always places green packets in their queue (possibly dropping previously queued yellow packets, so as to maintain the occupancy constraint). In the delivery substep, the algorithm gives the green queue strict priority over the yellow queue. Namely, this algorithm is allowed to violate the requirement to transmit packets in FIFO order. The threshold for the threshold algorithm was set to be the amount of additional buffer space available to the algorithm, namely, $T = \varepsilon B$. When the queue fills up with εB packets or more, yellow packets are not admitted to the queue.

A. Traffic Generation and Setup

Although widely used in various queuing environments, Poisson traffic is rather smooth and does not pose a challenge for AQM. We consider two scenarios. In the first we use a Markov modulated Poisson process (MMPP) with two states, `on` and `off`, with symmetric transition rates, which is more bursty than Poisson. During the `on` stage packets are generated with a rate of $\lambda_{\texttt{on}}$, which results in an effective rate across both `on` and `off` states of $\lambda_{\texttt{on}}/2$ (the effective rate is half the `on` rate since the transition rates are chosen to be symmetric). Traffic generated using the MMPP generator is colored by a token-bucket coloring module, according to the

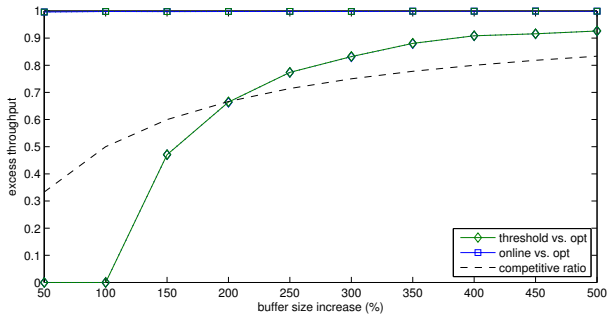

Fig. 5. The effect of buffer size increase for the case where no yellow packets are added, and overall yellow packets are ∼30% of the traffic. The dashed line represents the competitive ratio guarantee of our online algorithm.

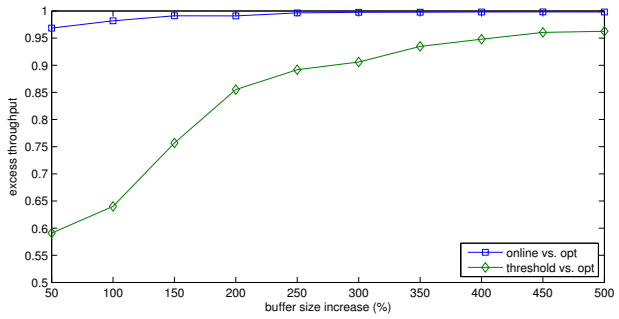

Fig. 6. The effect of buffer size increase for the case where yellow packets are added, and overall yellow packets are 40% of the traffic.

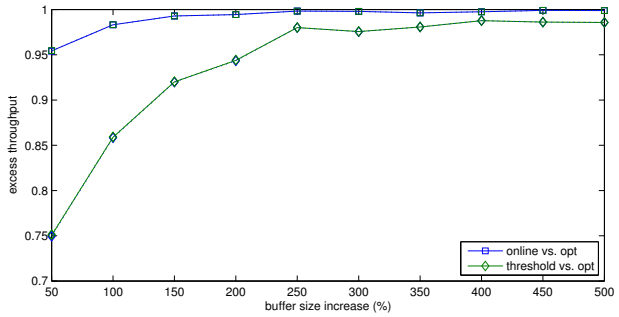

Fig. 7. The effect of buffer size increase for the case where yellow packets are added, and overall yellow packets are 50% of the traffic.

committed token-bucket parameters. Note that although this process indeed produces bursty traffic as required, excess traffic only appears at the end of a bursty period. In addition, even for high arrival rates (e.g., for $\lambda_{\mathrm{on}} > 2$, which implies an average arrival rate greater than 1), MMPP generated traffic produces ≈30% yellow traffic.

The second scenario captures the case where the traffic is an aggregate of flows, some of which are only best effort. This is modeled by an additional stream of excess traffic, whose arrival is governed by a Poisson process. This stream represents SLAs with zero committed traffic. The overall traffic in such cases consists of the interleaving of both the MMPP generated stream, and the excess Poisson generated stream. This enables us to both have excess traffic arrive not only at the tail of a bursty period, as well as have yellow traffic consist a larger fraction of the overall traffic. We note that it is common to assume that providers do not commit themselves to more than half their available bandwidth, thus the above approach enables us to explore scenarios where yellow traffic consists of roughly 50% of the traffic.

In all the results presented below, we conducted 10 rounds of simulation for each case considered, where each round consisted of simulating the arrival of 5000 packets, and using the committed buffer size parameter of $B = 20$, and rate parameter of $r = 1$ (this can be viewed as a merely normalizing factor). We further assume that there is no speedup, i.e., $s = r$. The plots presenting our results depict the average normalized throughput obtained in the simulations.

B. Simulation Results

Figures 5-7 depict the results comparing the throughput of both our proposed online algorithm, and the threshold algorithm, under 3 distinct yellow traffic intensities, as a function of the amount of additional buffer space available to the algorithm.

Figure 5 shows the normalized throughput of both algorithms as a function of the amount of additional buffer space, where traffic was generated by an MMPP without any additional yellow traffic. Figure 5 also depicts the performance guarantee of our online algorithm, as implied by Theorem 2.4, which serves as a lower bound on the throughput of our algorithm. We note that this lower bound is with comparison to the *actual implicit* optimal throughput possible, and not just compared to the upper bound on the optimal throughput. Recall that throughput is in terms of yellow packets only, since all algorithms must deliver all green packets. Our online algorithm matches the upper bound throughput for all levels of increased buffer size. The threshold algorithm fails to use the extra buffer at all for buffer increase of 100% and less. This is due to the fact that yellow packets always come at the end of a burst, where the buffer already contains B green packets and thus the buffer fills up above the threshold, causing all yellow packets to be rejected upon arrival. The threshold algorithm reaches 80% throughput only for a buffer increase of 300% and grows up to 90% for an increase of 500%. We note that for the case where traffic consists merely of the MMPP generated stream, our online algorithm does not drop yellow packets due to violation of the lag property. This is due to the fact that yellow packets only arrive at the end of a burst. As is apparent from the simulation results, our algorithm actually performs better than the minimum guarantee implied by the competitive ratio proved in Theorem 2.4.

Figure 5 also shows a plot of the guarantee provided by Theorem 2.4. Note that this plot is normalized by the actual value of OPT and not the upper bound on OPT used in normalizing our simulation results. For the traffic pattern we

simulated, our algorithm achieves much better throughput than this guarantee.

Figures 6 and 7 present the results for traffic which is an interleaving of an MMPP generated stream, and a Poisson generated stream consisting solely of yellow packets. Figure 6 presents the results where the overall number of yellow packets is 40% of the entire traffic, whereas Figure 7 presents the results where the overall number of yellow packets is 50% of the entire traffic. In both cases, the resulting traffic serves as a more challenging scenario for our algorithm, since in this case yellow packets distribution is not as predictable.

One of the effects of adding the additional yellow traffic is that our online algorithm no longer obtains the optimal throughput. In addition, for these traffic patterns our algorithm actively drops yellow packets from its queue in order to preserve the lag property. The performance of the threshold algorithm improves for these scenarios since now it can accept yellow packets that arrive not as part of a burst and thus have a chance to find the buffer below the threshold even for less than 100% buffer increase. For buffer sizes of 200% and below the difference between the online algorithm and the threshold algorithm are still substantial, however for larger buffer sizes the differences diminish. In general the higher the additive yellow traffic and the higher the buffer increase the difference between the algorithms' throughput decreases. However the 100-200% range is the one that seems the most suitable in practice (namely doubling to tripling of the buffer space required for committed traffic) and in this range the advantage demonstrated by our algorithm is noticeable for all the tested scenarios.

IV. Extensions and Conclusion

In this paper we studied a simple variant of the oldest question: what to do in the face of temptation. Specifically, we considered the problem of buffer management in the case of heterogeneous traffic, that contains both packets with guaranteed delivery as well as packets that generate cash reward for delivery. We adopted the conservative approach that assigns much more value to the potential penalty for violating a commitment, but, as we show, we could still enjoy much of the maximal possible profit, at the expense of requiring a little more resources. We show this analytically, by proving almost tight upper bounds on the competitive ratio of our algorithms. We further established that for traffic which is comprised of a mixture of bursty committed flows and best effort flows our algorithm performance is even better than our analytical guarantees.

We believe that our algorithms are practical and may be quite useful. Unlike the push-out algorithm of Cidon et al. [13] which requires dropping packets from arbitrary positions in the queue, we only need the ability to drop packets from the tail of the queue, a task that requires just an extra pointer per queue. The ease of implementation, as well as the analytical guarantees of performance seem quite promising. However, not all questions are solved. E.g., the upper and lower bounds on the competitive ratio are tight only for small additional buffer space. It seems interesting to close the gap for $\epsilon \geq 1$.

Acknowledgment

Research supported in part by the Next Generation Video (NeGeV) consortium, Israel. This work was done while the second author was with Tel Aviv University.

References

[1] S. Sathaye, "ATM forum traffic management specification version 4.1," ATM Forum 95-0013, Dec. 1995.

[2] S. Blake, D. Black, M. Carlson, E. Davies, Z. Wang, and W. Weiss, "An architecture for differentiated services," Internet RFC 2475, Nov. 1998.

[3] J. Heinanen and R. Guerin, "A two rate three color marker," Internet Engineering Task Force, Sep. 1999.

[4] K. Nichols and B. Carpenter, "Definition of differentiated services per domain behaviors and rules for their specification," Apr. 2001, internet RFC 3086.

[5] D. Grossman, "New terminology and clarifications for diffserv," Apr. 2002, internet RFC 3260.

[6] F. L. Faucheur, L. Wu, B. Davie, S. Davari, P. Vaananen, R. Krishnan, P. Cheval, and J. Heinanen, "Multi-protocol label switching (mpls) support of differentiated services," Internet RFC 3270, May 2002.

[7] M. E. Forum, "Ethernet services attributes phase 1," Metro Ethernet Technical Specifications, Nov. 2004.

[8] R. Santitoro, "Bandwidth profiles for ethernet servicesf," Metro Ethernet Forum White Paper, Jan. 2004.

[9] D. Sleator and R. Tarjan, "Amortized efficiency of list update and paging rules," *Communications of the ACM*, vol. 28, no. 2, pp. 202–208, 1985.

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

[11] S. Bergida and Y. Shavitt, "Analysis of shared memory priority queues with two discard levels," *IEEE Network*, vol. 21, no. 4, pp. 46–50.

[12] Y. Huang, R. Guérin, and P. Gupta, "Supporting excess real-time traffic with active drop queue," *IEEE/ACM Transactions on Networking*, vol. 14, no. 5, pp. 965–977, 2006.

[13] I. Cidon, R. Guérin, and A. Khamisy, "On protective buffer policies," *IEEE/ACM Transactions on Networking*, vol. 2, no. 3, pp. 240–246, 1994.

[14] A. Kesselman, Z. Lotker, Y. Mansour, B. Patt-Shamir, B. Schieber, and M. Sviridenko, "Buffer overflow management in qos switches," *SIAM Journal on Computing*, vol. 33, no. 3, pp. 563–583, 2004.

[15] M. Englert and M. Westermann, "Lower and upper bounds on fifo buffer management in qos switches," in *Proceedings of the 14th Annual European Symposium on Algorithms (ESA)*, 2006, pp. 352–363.

[16] Y. Mansour, B. Patt-Shamir, and O. Lapid, "Optimal smoothing schedules for real-time streams," *Distributed Computing*, vol. 17, no. 1, pp. 77–89, 2004.

[17] N. Andelman, Y. Mansour, and A. Zhu, "Competitive queueing policies for qos switches," in *Proceedings of the 14th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, 2003, pp. 761–770.

[18] S. Sahu, P. Nain, C. Diot, V. Firoiu, and D. F. Towsley, "On achievable service differentiation with token bucket marking for tcp," in *Proceedings of the ACM SIGMETRICS 2000 International Conference on Measurement and Modeling of Computer Systems*, 2000, pp. 23–33.

[19] Y. Chait, C. V. Hollot, V. Misra, D. F. Towsley, H. Zhang, and Y. Cui, "Throughput differentiation using coloring at the network edge and preferential marking at the core," *IEEE/ACM Transactions on Networking*, vol. 13, no. 4, pp. 743–754, 2005.

[20] S. Floyd and V. Jacobson, "Random early detection gateways for congestion avoidance," *IEEE/ACM Transactions on Networking*, vol. 1, no. 4, pp. 397–413, 1993.