



ELSEVIER

Available at
www.ComputerScienceWeb.com
POWERED BY SCIENCE @ DIRECT®

Computer Networks 42 (2003) 481–492

COMPUTER
NETWORKS

www.elsevier.com/locate/comnet

Nearly optimal FIFO buffer management for two packet classes [☆]

Zvi Lotker, Boaz Patt-Shamir ^{*}

Department of Electrical Engineering, Faculty of Engineering, Tel Aviv University, Tel Aviv 69978, Israel

Received 16 May 2002; received in revised form 27 January 2003; accepted 27 January 2003

Responsible Editor: J. Roberts

Abstract

We consider a FIFO buffer with finite storage space. An arbitrary input stream of packets arrives at the buffer, but the output stream rate is bounded, so overflows may occur. We assume that each packet has value which is either 1 or α , for some $\alpha > 1$. The buffer management task is to decide which packets to drop so as to minimize the total value of lost packets, subject to the buffer space bound, and to the FIFO order of sent packets. We consider push-out buffers, where the algorithm may eject packets from anywhere in the buffer. The best lower bound on the competitive ratio of on-line algorithms for buffer management is approximately 1.28. In this paper we present an on-line algorithm whose competitive ratio is approximately 1.30 for the *worst case* α . The best previous general upper bound was about 1.888.

© 2003 Elsevier Science B.V. All rights reserved.

Keywords: Buffer overflows; Competitive analysis; Quality of Service; Throughput; Classes of service

1. Introduction

Buffers can be found in almost all computer systems: they serve as a basic coupling component that enables communication without rigid synchronization. Packets enter with one traffic characteristic, and leave with another. The existence and importance of buffers is more pronounced in

data communication networks, where buffers are found essentially in each connection point: a computer's network adapter (NIC), a switch's interface (port), etc. In most settings the buffers are required to adhere to FIFO ordering as part of the correctness specifications.

In many cases, the traffic into and out of the buffer obeys certain known restrictions that allow the designer to choose a buffer that will accommodate all possible scenarios (e.g., *leaky bucket constrained* traffic [6]). In many other cases, however, incoming traffic does not have a deterministic upper bound, or, equivalently, the only upper bounds known require more resources than available. In these cases a *buffer management policy* is called for to handle overflow events. The simplest

[☆] A preliminary version of the paper appeared in Proceedings of the 21st ACM Symposium on Principles of Distributed Computing, 2002.

^{*} Corresponding author. Tel.: +972-3-640-7036; fax: +972-3-640-5413.

E-mail addresses: zvilo@eng.tau.ac.il (Z. Lotker), boaz@eng.tau.ac.il (B. Patt-Shamir).

and most popular approach to overflow management is “tail drop”: new packets are dropped if there’s no room in the buffer. If all packets are equally important, this policy is good enough. The situation is more interesting when different packets have different *values*, as is the case when different levels of service are to be supported. Let us give two basic examples for different packet values. First, there is the obvious scenario where each delivered packet has a cash value: In the Internet, many pricing mechanisms have been proposed (see, for example, [5,8,14] and references therein), and one of the basic approaches to pricing is a per-packet fee. In the case of two levels of service, we may assume that we have two packet prices: a “regular” packet of value 1, and a “valuable” packet of value $\alpha > 1$. Another scenario where a two-value model seems to make sense is in the context of constrained incoming streams: For example, in ATM some incoming streams commit to a limiting traffic envelope. Packets—called “cells” in this context—violating the constraint are marked (using the cell loss priority bit). Since it is preferable to deliver even violating packets if possible, we may assume that a packet complying with its traffic descriptor has some intrinsic value $\alpha > 1$, and other (violating) packets have value 1, where the parameter $\alpha > 1$ represents the “strictness” of the system.

In this paper, we analyze a simple abstraction of a buffer, that can be roughly described as follows. We are given a buffer that can hold at most B packets. In each time step, an arbitrary set of packets arrives at the buffer, and at most one packet may leave the buffer. Each packet p has value $v(p) \in \mathbb{R}^+$. We concentrate on the special case where packets may have only two values: 1 and $\alpha > 1$. The buffer management algorithm decides which packets to drop from the buffer and which packets to send. At each step, any packet from among those currently stored in the buffer and from among the newly arriving packets may be dropped (this is the *push out* buffer model). FIFO order must be maintained over the sent packets, in the sense that if p arrived before q and both are sent, then p is sent before q . (Note that FIFO buffers ensure bounded delivery time for packets that are not dropped.)

The goal of the algorithm is to maximize the total value of delivered packets. We use *competitive analysis* [3,17] to evaluate algorithm performance. Specifically, the *competitive ratio* of an algorithm alg is an upper bound, over all possible arrival sequences, on the ratio of the value sent by an optimal (off-line) algorithm to the value sent by alg .

Let us summarize briefly some results directly relevant to our work. First, note that if packet values are in the range $[1, \alpha]$, then any work-conserving policy that does not drop packets while there’s room in the buffer (including the tail-drop policy) is α -competitive.¹ This is because all these algorithms send the maximal possible *number* of packets. It is straightforward to see that in some cases, tail-drop actually sends only $1/\alpha$ value of the value sent by the optimal algorithm. On the other hand, it is known that no deterministic on-line algorithm can have competitive ratio smaller than 1.28 [15,18]. The lower bound is proved using two packet values. The most natural buffer management policy is the *greedy* policy, that drops the cheapest packets when an overflow occurs. Mansour et al. [15] give a relatively simple proof that the greedy policy is 4-competitive. Kesselman et al. [9] give a much more subtle proof that shows that the competitive ratio of the greedy policy is in fact $2 - 2/(\alpha + 1)$, for any packet values in the range $[1, \alpha]$. It is also shown in [9] that the “greedy head-drop” policy (the greedy algorithm which prefers dropping old packets in case of a tie) is the best greedy policy. For the model of two possible values $\{1, \alpha\}$, Kesselman and Mansour [10] propose a more “proactive” algorithm with competitive ratio $\sqrt{\alpha}/(\sqrt{\alpha} - 2)$ for $\alpha > 4$. Combining the results of the greedy algorithm with the latter, one gets an algorithm with worst-case competitive ratio about 1.888 for any α in the two-value case.

Our results. In this work, we significantly reduce the competitive ratio of buffer management for the two packet values model. We do it with a new algorithm, whose competitive ratio is

¹ An algorithm is called work-conserving if it always sends a packet if there’s one available.

$1 + 1/\sqrt{\alpha} + O(1/\alpha)$, and never more than 1.30 for any α . The algorithm is *memoryless*, i.e., its action depends only on the current state of the buffer, and no additional persistent state is required.

More about related work. Many research papers deal with packet drop policies in communication networks—see, for example, the survey of [11] and references therein. Some drop mechanisms, such as RED [7], are designed to signal congestion to the sending end. The approach abstracted in our model, where packets have values and the goal is to maximize the total throughput value, is implicit in DiffServ [2,4] and ATM [19].

There has been work on analyzing various aspects of the model using classical queuing theory, and assuming Poisson arrivals [16]. The Poisson arrival model has been seriously undermined by the discovery of the heavy tail nature of data traffic [12] and the chaotic nature of TCP [20].

Another model studied in [1] assumes that one cannot discard a packet already in the buffer, and thus the algorithm may only control admission into the buffer. Aiello et al. [1] give tight bounds on the competitive factor of various algorithms for the two value model. In [15], the question of video smoothing is studied. Among the results in that paper, they prove an upper bound of 4 on the competitive ratio of the greedy algorithm and a lower bound of 1.25 on the ratio of any on-line algorithm (the lower bound was later improved to about 1.28 [13,18]). The work of [10] studies competitive ratio of the lost value rather than the throughput value we use here.

Paper organization. The remainder of this paper is organized as follows. In Section 2 we define the model and the notation we use. In Section 3 we specify our on-line algorithm *mf* and a reference optimal algorithm. In Section 4 we analyze the competitive factor of *mf*. We conclude with a few comments in Section 5.

2. Model and notation

In this section we formalize the model and the notation we use. We assume a discrete time model, i.e., time progresses in *steps* numbered 1, 2, 3, etc. (see Fig. 1).

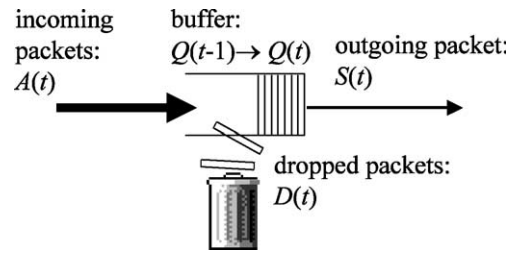


Fig. 1. Schematic representation of the model. The buffer contains at most B packets.

2.1. Arrival sequences

In each time step $t \geq 1$, a set $A(t)$ of packets arrive. For a packet p , $T(p) = t$ iff $p \in A(t)$, i.e., $T(p)$ is the *time of arrival* of p . Each packet $p \in A(t)$ has a *value* $v(p) \in \mathbb{R}^+$. For the most part of this paper, we assume that $v(p) \in \{1, \alpha\}$ for some $\alpha > 1$. We assume that each packet has a distinct *index* $i(p) \in \mathbb{N}$, such that if packet p arrives before packet q then $i(p) < i(q)$. The *duration* of an arrival sequence A is the number of steps until no more packets arrive (we assume that all arrival sequences A satisfy $A(1) \neq \emptyset$, except for the empty arrival sequence whose duration is 0). The *size* of an arrival sequence A , denoted $|A|$, is the total number of packets that arrive.

2.2. Buffers

In each time step t , there is a set $Q(t)$ of packets referred to as the *buffer*. $|Q(t)|$ is called the *buffer occupancy* at time t . We say that packet $p \in Q$ is *above* packet $q \in Q$ in the buffer if $i(p) > i(q)$; the *below* relation is defined analogously. The packet with the minimal index in Q is said to be at the *bottom* of the buffer, or at the *head* of the queue. When all packets have value either 1 or α , we use $\text{exp}(Q(t)) \stackrel{\text{def}}{=} |\{p \in Q(t) : v(p) = \alpha\}|$, i.e., $\text{exp}(Q(t))$ denotes the number of expensive packets in $Q(t)$.

2.3. Algorithms

The buffer management algorithm decides, in each step t , which of the packets in $Q(t-1) \cup A(t)$ are to be sent out, and which are dropped, so that $|Q(t)| \leq B$ for some given parameter B called the

buffer size. Formally, the packets sent at time t are denoted $S(t)$ and the packets dropped at time t are denoted $D(t)$. If $p \in D(t)$ we say that p was *discarded* at time t . The sequence $S(1), S(2), \dots$ is called a *schedule*. We sometimes abuse notation and refer to the sets of packets $\bigcup_i A(t)$, $\bigcup_i S(t)$ and $\bigcup_i D(t)$ by A , S and D , respectively. Note that Q , S , and D are functions of time, arrival sequence, and algorithm. Usually the arrival sequence and algorithm are clear by the context, but in other cases we use superscripts to denote the arrival sequence and subscripts to denote the algorithm, as in $S_{\text{alg}}^A(t)$.

If $S(t)$ and $D(t)$ are functions of $A(1), \dots, A(t)$ only, then the algorithm is said to be *on-line*, and otherwise it is *off-line*. If $S(t)$ and $D(t)$ are functions of $Q(t-1) \cup A(t)$ only, then the algorithm is said to be *memoryless*.

The main restrictions we assume are as follows:

- For all t , $|S(t)| \leq R$ for some parameter R called the *drain rate*. In the remainder of this paper, we assume for simplicity that $R = 1$, i.e., at most one packet can be sent in each step.
- The algorithms are *work conserving*, i.e., they always send a packet if there is one available and if there is available bandwidth. Formally, an algorithm is called work-conserving if for all t , $|S(t)| = \min(R, |Q(t-1) \cup A(t)|)$.
- We assume that for all t , $\max\{i(p) : p \in S(t)\} < \min\{i(p) : p \in Q(t)\}$, i.e., a packet cannot remain in the buffer after a packet of larger index was sent. That is the FIFO restriction.

2.4. Total value and competitive ratios

The value of a set of packets S is defined by $v(S) \stackrel{\text{def}}{=} \sum_{p \in S} v(p)$. The value of an algorithm alg on a given arrival sequence A is defined by $v_{\text{alg}}(A) \stackrel{\text{def}}{=} \sum_{t \geq 1} v(S_{\text{alg}}^A(t))$. The *competitive ratio of an algorithm* alg on an arrival sequence A is defined by

$$\text{cr}_{\text{alg}}(A) \stackrel{\text{def}}{=} \frac{\max\{v_{\text{off}}(A) : \text{off is an off-line algorithm}\}}{v_{\text{alg}}(A)}.$$

Note that the competitive ratio is at least 1 for any algorithm and any arrival sequence. The *competitive ratio of an algorithm* alg is

$$\text{cr}_{\text{alg}} \stackrel{\text{def}}{=} \sup\{v(A)_{\text{alg}} : A \text{ is an arrival sequence}\}.$$

3. Algorithms

In this section we present our reference off-line algorithm called *opt* and our proposed on-line algorithm called *mf*. Before we start, we define the concept of replaceability which plays a central role in our analysis.

Informally, a packet p is said to be replaceable by a packet q if by dropping p and adding q to the schedule, the buffer size is not changed. Formally, we have the following definition.

Definition 3.1. Let A, S and D be arrival, send and drop sequences, respectively. Let $p, q \in A$ be packets such that $p \in S$ and $q \in D$. We say that p is replaceable by q in S if one of the following conditions hold:

1. $i(p) < i(q)$, and $|Q(t)| > 0$ for all time steps $T(p) \leq t < T(q)$.
2. $i(p) > i(q)$, and $|Q(t)| < B$ for all time steps $T(q) \leq t < T(p)$.

Lemma 3.1. Suppose that for some arrival sequence A we have that a packet $p \in A$ is replaceable by a packet $q \in A$ in a work-conserving schedule S with buffer size B . Then the work-conserving schedule S' resulting from S by discarding p when it arrives and accepting q has buffer requirement B .

Proof. Suppose first that $i(p) < i(q)$. Clearly, buffer occupancy in S' is identical to S for all times before $T(p)$. In the time interval $[T(p), T(q) - 1]$, buffer occupancy in S' is exactly one less than the corresponding occupancy in S , since the buffer in S was never empty in that interval. Also, from time $T(q)$ onwards, buffer occupancy in both buffers is the same. The case $i(p) > i(q)$ is dual, noting that now we have that buffer occupancy of S' in the time interval $[T(q), T(p) - 1]$ is one more than the corresponding occupancy in S . \square

3.1. An optimal off-line algorithm: *opt*

We now spell out a specific off-line algorithm, called “*opt*”, that will serve us as a reference optimal solution. This concretization allows us to

prove a few properties we use in the analysis of our on-line algorithm. From now on, whenever we refer to the optimal schedule, we refer to the unique schedule produced by *opt*.

The basic idea in *opt* is that the problem of overflow management has a matroid structure for each given arrival sequence (see [9] for details). This property guarantees that the “greedy” algorithm for matroids finds an optimal solution. Our reference algorithm, *opt*, is just the greedy algorithm with a specific order on packets with equal values: if two packets have the same value, the older packet takes precedence. This makes the resulting schedule unique. Pseudo-code for *opt* is given in Fig. 2. We remark that the feasibility testing of line 4 can be done by running the schedule and finding whether its maximal buffer size exceeds B (more efficient ways are possible).

The schedule produced by *opt* has the following simple properties, which we use extensively.

Lemma 3.2. *Fix an arrival sequence A . If p is replaceable by q in S_{opt} and $v(p) = v(q)$, then $i(p) < i(q)$.*

Proof. Suppose for contradiction that $i(q) < i(p)$. Since p and q have the same value, the optimal algorithm considers q before p . Let S be the current schedule considered by the algorithm when q is considered. Since p is replaceable by q , we have that the buffer of S (whose occupancy is never more than the occupancy in the full schedule) is never full in $[T(q), T(p) - 1]$, and since p is included in the final optimal schedule, it follows that q could have been entered too, a contradiction. \square

```

sort all packets in decreasing value,
  and within each value class, by increasing index.
 $S \leftarrow$  empty schedule
while the list is non-empty do
   $p \leftarrow$  head of the sorted list
  if  $S \cup \{p\}$  is feasible then  $S \leftarrow S \cup \{p\}$ ;
  remove head of the list
output  $S$ 

```

Fig. 2. Pseudo-code for algorithm *opt*.

Corollary 3.3. *An expensive packet p is not accepted by *opt* if and only if the buffer of *opt* is full with expensive packets at time $T(p)$.*

The following straightforward property is common to all optimal algorithms.

Observation 1. Let A be an arrival sequence, and let A' be an arrival sequence defined by adding one packet to A . Then $v_{\text{opt}}(A') \geq v_{\text{opt}}(A)$.

3.2. The on-line algorithm: *mf*

Our on-line algorithm is called *mark&flush*, abbreviated *mf*. Given a non-negative parameter $r \geq 0$, the *mf^r* algorithm is as follows (see pseudo-code in Fig. 3). For each packet p in its buffer, the algorithm maintains a label $n(p) \in \mathbb{N} \cup \{\perp\}$. When $n(p) = \perp$ we say that p is *unmarked*, and when $n(p) = i$, then p is said to be *marked* by packet q , where $i(q) = n(p)$.

The action of the algorithm in a step is described by two phases. In the first phase, overflows are resolved using the “greedy tail-drop” rule: cheap packets are dropped first, and within each value class, older packets are discarded before newer packets. Packets dropped in the first phase are called *overflow packets*.

In the second phase, the algorithm looks at the newly admitted expensive packets, from the bottom up. Each packet p marks the r closest cheap packets below p which are not yet marked: Marking of a packet q is done by setting $n(q) \leftarrow i(p)$. (No packet is marked if there is no cheap unmarked packet below p .) Then the algorithm examines the packet p_0 at the head of the queue. If p_0 is unmarked, it is sent, and the algorithm terminates. Otherwise, all packets with marks smaller or equal to $n(p_0)$ are discarded, the first remaining packet is sent, and the algorithm terminates. Packets discarded in the second phase of the algorithm are said to be *preempted*.

The value of r is specified later, as a function of α . For simplicity of presentation, we assume now that r is integral; we explain later how to extend the algorithm to any $r \in \mathbb{R}$.

We state a few simple properties of the algorithm that will become handy shortly (see Fig. 4).

<pre> if $A(t) + Q(t-1) > B + 1$ then sort $A(t) \cup Q(t-1)$ by increasing value, and within each value class, by decreasing index $Q(t) \leftarrow$ first $B + 1$ elements of the sorted list for $k \leftarrow 1$ to B do if $v(Q(t)[k]) = \alpha$ and $T(Q(t)[k]) = \text{now}$ then $c \leftarrow r$; $j \leftarrow k - 1$ while $c > 0$ and $j > 0$ do if $v(Q(t)[j]) = 1$ and $n(Q(t)[j]) = \perp$ then $n(Q(t)[j]) \leftarrow i(Q(t)[k]);$ $c \leftarrow c - 1$; $j \leftarrow j - 1$ $\ell \leftarrow n(Q(t)[0])$ if $\ell \neq \perp$ then for each $p \in Q(t)$ do if $n(p) \neq \perp$ and $n(p) \leq \ell$ then remove p from $Q(t)$ send packet from head of the queue </pre>	<p style="text-align: right;"><i>phase 1: overflow resolution</i></p> <p style="text-align: right;"><i>greedy tail-drop rule</i></p> <p style="text-align: right;"><i>phase 2: preemption</i></p> <p style="text-align: right;"><i>scan from the bottom up</i></p> <p style="text-align: right;"><i>new expensive packet: try to mark</i></p> <p style="text-align: right;"><i>scan from $k - 1$ downwards</i></p> <p style="text-align: right;"><i>cheap unmarked packet: mark</i></p> <p style="text-align: right;"><i>discard packets with mark smaller than first mark</i></p>
--	---

Fig. 3. Pseudo-code for algorithm mf with parameter r , as run at time t . $Q(t)[k]$ is the k th packet from the head of the buffer.

Lemma 3.4. *If mf preempts a packet in step t , then $v(S_{mf}(t)) = \alpha$.*

Proof. Since marking is done top-down, it follows that if p is marked, then all packets between p and $n(p)$ are marked (by $n(p)$ or by packets with

smaller index). Therefore, if a cheap packet is dropped from the head of the queue, then there must be an expensive packet that will be sent out. \square

The next lemma follows from a straightforward induction on time.

Lemma 3.5. *If $p \in Q_{mf}(t)$ is an expensive packet, and $|\{q: n(q) = p\}| < r$, then all cheap packets below p are marked.*

The argument below is similar in spirit to [10].

Lemma 3.6. *For all t , if $\text{exp}(Q_{mf}(t)) \geq B/(r + 1)$, then $v(S_{mf}(t)) = \alpha$.*

Proof. There are two cases to consider. If each expensive packet $p \in Q_{mf}(t)$ has r packets p' with $n(p') = p$, then by simple algebra we have that all cheap packets in $Q_{mf}(t)$ are marked, and hence the next packet to be transmitted is expensive. Otherwise, there exists at least one expensive packet p_0 such that $|\{p': n(p') = p_0\}| < r$. In this case, by Lemma 3.5, all cheap packets in $Q_{mf}(t)$ that arrived before p_0 are marked, and therefore the next packet to be sent is necessarily expensive. \square

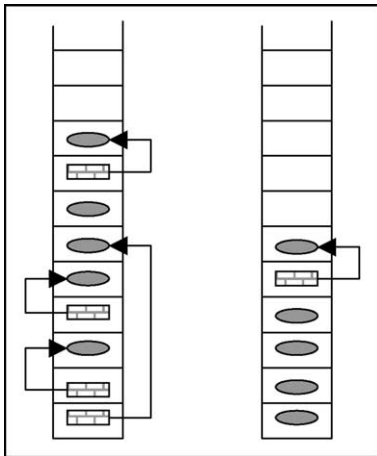


Fig. 4. Preemption with $r = 1$. Cheap packets are square, expensive packets are oval. An arrow from a cheap packet p to an expensive packet q means that p is marked by q , i.e., $n(p) = i(q)$. Left: before preemption. Right: after preemption.

4. Competitive analysis

In this section we prove our main result, namely that the competitive ratio of mf is about 1.30. Our strategy in analyzing the competitive ratio of mf is to use induction on the number of packets in the arrival sequence. Informally, given an arrival sequence, we apply a series of simplifying transformations while preserving the competitive ratio. Each transformation strictly decreases the size of the arrival sequence, allowing us to invoke the induction hypothesis. Eventually, the arrival sequence boils down to one of two possible scenarios (depending on the value of the first packet sent by opt), which we analyze directly.

We start with a simple lemma.

Lemma 4.1. *For any arrival sequence A there exists an arrival sequence A' such that $|A'(1)| \leq B + 1$ and $\text{cr}(A') \geq \text{cr}(A)$.*

Proof. If $|A(1)| \leq B + 1$ then $A' = A$ and we are done. Otherwise, define $A'(1)$ to be the packets not dropped by mf at step 1, and $A'(t) = A(t)$ for all $t > 1$. Clearly, $v_{\text{mf}}(A') = v_{\text{mf}}(A)$. It remains to show that $v_{\text{opt}}(A') \geq v_{\text{opt}}(A)$: this follows directly from the specification of the optimal algorithm, that clearly accepts all first $B + 1$ expensive packets (if exist), and a subset of the first cheap packets in the remaining room. \square

By virtue of Lemma 4.1, we may assume without loss of generality that $|A(1)| \leq B + 1$, i.e., that there is no overflow in the first step.

We now define one of the main tools of our analysis: We show how to split a arrival sequence at some points into two, such that at least one of the parts preserves the competitive ratio.

Definition 4.1. Let A be an arrival sequence with duration t_e . A time step t is called a *splitting point* if $t < t_e$ and $|Q_{\text{mf}}(t)| = \exp(Q_{\text{mf}}(t)) = \exp(Q_{\text{opt}}(t)) = |Q_{\text{opt}}(t)|$.

In other words, t is a splitting point if it is not the last step, and at t , the buffers of mf and opt contain exactly the same number of expensive packets and no cheap packets.

Lemma 4.2. *Let A be an arrival sequence with splitting point t_0 . Then there exists an arrival sequence A' such that $|A'| < |A|$ and $\text{cr}(A') \geq \text{cr}(A)$.*

Proof. Let $b = \exp(Q_{\text{mf}}(t_0))$. By definition,

$$\text{cr}(A) = \frac{\sum_{t=1}^{t_0+b} v(S_{\text{opt}}(t)) + \sum_{t=t_0+b+1}^{\infty} v(S_{\text{opt}}(t))}{\sum_{t=1}^{t_0+b} v(S_{\text{mf}}(t)) + \sum_{t=t_0+b+1}^{\infty} v(S_{\text{mf}}(t))}.$$

We define two arrival sequences. $A_1(t) = A(t)$ for $t = 1, \dots, t_0$ and $A_1(t) = \emptyset$ for $t > t_0$. $A_2(1) = Q_{\text{mf}}(t_0 + b)$ and $A_2(t) = A(t_0 + b + t - 1)$ for $t > 1$. We claim that

$$\begin{aligned} v_{\text{mf}}(A) &= v_{\text{mf}}(A_1) + v_{\text{mf}}(A_2), \\ v_{\text{opt}}(A) &\leq v_{\text{opt}}(A_1) + v_{\text{opt}}(A_2). \end{aligned} \quad (1)$$

Proving Eq. (1) will complete the proof, since both A_1 and A_2 are shorter than A , and at least one of them has a larger competitive ratio.

To see that Eq. (1) is true, note that in steps $t_0 + 1, \dots, t_0 + b$, both algorithms send only expensive packets under A and A_1 , and therefore their value under A in the first $t_0 + b$ steps match exactly their values under A_1 .

Now consider A_2 . We show that $Q_{\text{opt}}^A(t_0 + b) \subseteq Q_{\text{mf}}^A(t_0 + b)$. This will show that $v_{\text{opt}}(A_2)$ is at least the value of opt on A in steps $t_0 + b$ and onwards. First we claim that for all $t \in [t_0, t_0 + b - 1]$, we have $\exp(Q_{\text{mf}}^A(t)) = \exp(Q_{\text{opt}}^A(t))$: this follows from the fact that both algorithms start with a buffer with b expensive packets, send only expensive packets in these steps, and therefore lose expensive packets due only to overflow, which are the same. Moreover, it follows that if $p \in Q_{\text{opt}}^A(t_0 + b)$ is an expensive packet, then $p \in Q_{\text{mf}}^A(t_0 + b)$: this is because p must have arrived in the time interval $[t_0 + 1, t_0 + b]$, and since there is room for it in opt, there is room for it in mf. To complete the proof that $Q_{\text{opt}}^A(t_0 + b) \subseteq Q_{\text{mf}}^A(t_0 + b)$, we show that if $p \notin Q_{\text{mf}}^A(t_0 + b)$ is a cheap packet, then $p \notin Q_{\text{opt}}^A(t_0 + b)$. For suppose not: then $p \notin Q_{\text{mf}}^A(t_0 + b)$ and $p \in Q_{\text{opt}}^A(t_0 + b)$. At the time that mf discards p , it resides in the buffer of opt, and hence there is room in the buffer of mf for a cheap packet. Since opt prefers cheap packets the way mf does, we get a contradiction. \square

Lemma 4.3. *Let A be an arrival sequence such that $v(S_{mf}(1)) = v(S_{opt}(1))$. Then there exists another input sequence A' such that $|A'| < |A|$ and $cr(A') \geq cr(A)$.*

Proof. We first show that both algorithms send the exact same packet at the first step, and then define A' that does not contain that packet.

We distinguish between two cases. If $v(S_{mf}(1)) = v(S_{opt}(1)) = 1$, then mf did not discard any packet by Lemma 3.4. Also, by the specification of opt we have that the cheap packet it sends must be the first packet too. Let this packet be denoted by p . We define $A'(1) = A(2) \cup A(1) \setminus \{p\}$, and $A'(t) = A(t+1)$ for all $t > 1$. Clearly, $|A'| = |A| - 1$, and it is straightforward to verify that both algorithms send the same packets except p under A and A' , namely $v_{opt}(A') = v_{opt}(A) - 1$ and $v_{mf}(A') = v_{mf}(A) - 1$.

If mf sends an expensive packet p at the first step, then using the specification of mf and opt, then both send the same first packet. However, mf may have discarded some packets ahead of p , but fortunately, opt dropped the same packets. So we may define $A'(1) = A(2) \cup A(1) \setminus (\{p\} \cup D_{mf}(1))$ and $A'(t) = A(t+1)$. Again, $|A'| < |A|$ and it is easy to verify that both algorithms send the same packets except p in A and A' . \square

4.1. First case: opt sends an expensive packet at the first step

We now analyze the case where opt sends an expensive packet at the first step. We start with the following lemma.

Lemma 4.4. *Suppose that $v(S_{opt}(1)) = \alpha$ and that $v(S_{mf}(1)) = 1$. Then there exists a time t_e such that*

- (1) *opt sends only expensive packets in steps $1, \dots, t_e$,*
- (2) $\exp(Q_{opt}(t_e)) = B$,
- (3) $\exp(Q_{mf}(t_e)) = B$.

Proof. Since by Lemma 4.3 we know that mf sends a cheap packet at the first step, we have that the first packet p_0 in the arrival sequence is cheap: this follows from the fact that mf did not drop any

packet in the first step. Now, if opt admits into the buffer any cheap packet p before it is full with expensive packets, then p_0 and p are replaceable by definition, contradiction to Lemma 3.2. This proves (1) and (2).

We now prove (3). Clearly, for any algorithm and any time step t , $\exp(Q(t))$ is the number of expensive packets that arrive, minus the number of expensive packets sent or dropped. Now, observe that opt sends an expensive packet at the maximal possible speed up to t_e , and that by Corollary 3.3, we have that opt does not drop an expensive packet before time t_e . Next, let t' be the first time that mf drops an expensive packet. Clearly, $\exp(Q_{mf}(t)) \geq \exp(Q_{opt}(t))$ for all $t \leq t'$. If $t' \geq t_e$, we are done. Otherwise, $t' < t_e$. Let t'' be the last time mf drops an expensive packet before time t_e . Clearly, $\exp(Q_{mf}(t'')) = B$, and from that time until t_e , the number of expensive packets sent by mf is at most the number of expensive packets sent by opt. It follows that $\exp(Q_{mf}(t)) \geq \exp(Q_{opt}(t))$ for all $t'' \leq t \leq t_e$, and we are done. \square

We now arrive at a key property of the scenario we consider: the number of expensive packets in both buffers is always close.

Lemma 4.5. *Suppose that $v(S_{opt}(1)) = \alpha$ and that $v(S_{mf}(1)) = 1$. Then for all time steps $t \leq t_e$,*

$$\exp(Q_{opt}(t)) \leq \exp(Q_{mf}(t)) \leq \exp(Q_{opt}(t)) + \frac{B}{r+1}.$$

Proof. By induction on time. The base case $t = 0$ is trivial. Suppose that lemma holds for time t , and consider step $t + 1$. The same number of expensive packets is made available to both algorithms in $A(t+1)$, and hence the only way for the difference $\exp(Q_{mf}(t)) - \exp(Q_{opt}(t))$ to change is if opt sends or rejects a different number of expensive packets than the number sent or rejected by mf. By Lemma 4.4, opt does not send a cheap packet. If opt rejects an expensive packets, then $\exp(Q_{opt}(t+1)) = B$ and then $\exp(Q_{mf}(t+1)) = B$ by induction. So suppose that opt sends an expensive packet and does not lose any expensive packet. If mf sends an expensive packet too, then the lemma holds by the induction hypothesis. If mf sends a cheap packet at

time $t + 1$, then by Lemma 3.6 it must be the case that $\exp(Q_{\text{mf}}(t + 1)) \leq B/(r + 1)$, and the lemma follows, since $\exp(Q_{\text{opt}}(t + 1)) \geq 0$. \square

Using the above results, we can prove another simplification step.

Lemma 4.6. *Let A be an unsplittable arrival sequence with $v(S_{\text{opt}}(1)) = \alpha$ and $v(S_{\text{mf}}(1)) = 1$. Then if there exist $t_1 < t_e$ such that $\exp(Q_{\text{mf}}(t_1)) = B$ and $\exp(Q_{\text{opt}}(t_e)) = B$, then there exists another arrival sequence A' such that $|A'| < |A|$ and $\text{cr}(A') \geq \text{cr}(A)$.*

Proof. By Lemma 4.4 and the definition of splitting points, the arrival sequence ends when opt is full with expensive packets for the first time. Call this time t_e . Suppose now that mf is full with expensive packets at some time $t_1 < t_e$. We define an arrival sequence A' as follows. $A'(t) = A(t)$ for all $t \neq t_1$, and $A(t_1)$ is $A(t)$ plus $(B - \exp(Q_{\text{opt}}^A(t_1)))$ new expensive packets (with index higher than all other packets arriving at time t_1). In other words, A' is defined by adding to A as many expensive packets as opt can accommodate at time t_1 . We first note that mf sends the same packets on A and A' : this follows from the assumption that mf is full with expensive packets at time t_1 , and will not admit the additional packets by its “tail-drop” preference. Next, we observe that opt will accept all the new packets by its preference to expensive packets. It follows that t_1 is a splitting point for A' . Finally, we note that while $v_{\text{mf}}(A') = v_{\text{mf}}(A)$ (because mf sends exactly the same packets in A and A'), we also have that $v_{\text{opt}}(A') \geq v_{\text{opt}}(A)$: this follows from Observation 1. Since t_1 is a splitting point we can use Lemma 4.2 and get that there exists an arrival sequence A'' such that $|A''| < |A'|$ and $\text{cr}(A'') \geq \text{cr}(A')$. In fact, looking closely at the proof of Lemma 4.2, it can be seen that $|A''| < |A|$; we omit the details here. \square

We now arrive at a case where we can directly compute the competitive ratio.

Lemma 4.7. *Let A be an unsplittable arrival sequence for which $v(S_{\text{opt}}(1)) = \alpha$ and $v(S_{\text{mf}}(1)) = 1$, and such that $\exp(Q_{\text{mf}}(t)) = B$ if and only if $\exp(Q_{\text{opt}}(t)) = B$. Then*

$$\text{cr}(A) = \frac{(2+r)\alpha}{1+\alpha(1+r)}.$$

Proof. By Lemma 4.4 and the definition of splitting points, we have opt sends only expensive packets, and since mf is work-conserving, mf sends a packet in each step $1, \dots, t_e$. By Lemma 4.4, we get that both mf and opt send the same number of packets under A . Let n_1 and n_x be the number of cheap packets and expensive packets, respectively, sent by mf under A . We have that

$$\text{cr}(A) \leq \frac{\alpha(n_1 + n_x)}{n_1 + \alpha n_x}. \quad (2)$$

By Lemma 4.4, we have that $n_x \geq B$. We now argue that $n_1 \leq B/(r + 1)$. To see that, note that for all $t < t_e$, we have that $\exp(Q_{\text{mf}}(t)) - \exp(Q_{\text{opt}}(t))$ is exactly the number of cheap packets sent by mf up to time t : the same number of packets arrive at both queues, and no expensive packets are discarded by the condition that $\exp(Q_{\text{mf}}(t)) < B$ for all $t < t_e$. Since $\exp(Q_{\text{mf}}(t)) - \exp(Q_{\text{opt}}(t)) \leq B/(r + 1)$ by Lemma 4.5, we have that $n_1 \leq B/(r + 1)$. By Eq. (2), the bound on $\text{cr}(A)$ is maximized when n_1 is maximized. Thus we get

$$\begin{aligned} \text{cr}(A) &\leq \frac{\alpha\left(\frac{B}{r+1} + B + 1\right)}{\frac{B}{r+1} + \alpha(B+1)} < \frac{\alpha\left(\frac{B}{r+1} + B\right)}{\frac{B}{r+1} + \alpha B} \\ &= \frac{(2+r)\alpha}{1+\alpha(1+r)}. \quad \square \end{aligned}$$

Note that the ratio in Lemma 4.7 is tight only for $B \rightarrow \infty$.

4.2. Second case: opt sends a cheap packet at the first step

We now consider arrival sequences in which opt sends a cheap packet in the first step. Let t_z denote the first time the buffer of opt is empty.

Lemma 4.8. *If $v(S_{\text{opt}}(1)) = 1$, then in all time steps $1, \dots, t_z$, opt accepts all expensive packets.*

Proof. Clearly opt sends the first packet p_1 in the arrival sequence. By definition, p_1 is replaceable by

any packet that arrives in the time interval $[1, t_z]$. Since $p_1 \in S_{\text{opt}}$, we have from Lemma 3.1 that opt did not discard any expensive packet. \square

The following is the key lemma for this case.

Lemma 4.9. *Let A be an arrival sequence such that $v(S_{\text{opt}}(1)) = 1$, $v(S_{\text{mf}}(1)) = \alpha$ and opt does not accept some packet that arrives in the time interval $[1, t_z]$. Then there exists an arrival sequence A' such that $|A'| < |A|$ and $\text{cr}(A') \geq \text{cr}(A)$.*

Proof. Let p_1, p_2 be the first cheap packet in $A(1)$, and the last cheap packet in $D_{\text{mf}}^A(1)$ respectively. By definition of p_2 , after the arrival of p_2 an expensive packet arrives. This expensive packet is the first in a succession of expensive packets. Denote the last expensive packet in this succession by p_3 . Note that if $i^A(p_2) < i^A(p) < i^A(p_3)$ then $v(p) = \alpha$. Let p_4 be the first cheap packet opt drops in the time interval $[1, t_z]$. By the assumption of the lemma, the packets p_1, p_2, p_3, p_4 are well defined.

Next we define a new arrival sequence $A'(1) = A(1) \setminus \{p_2\}$, and $A'(t) = A(t)$ for all $t > 1$. Clearly $|A'| < |A|$. We will show that $v(S_{\text{opt}}^{A'}) = v(S_{\text{opt}}^A)$ and $v(S_{\text{mf}}^{A'}) = v(S_{\text{mf}}^A)$.

We start with opt. If p_2 is dropped by opt in A , our claim is trivial. Assume p_2 is sent by opt. In this case we get that p_2 is replaceable by p_4 since the buffer of opt is not empty until time t_z and since opt prefers old packets.

It remains to prove that total value of the packets sent by mf is equal on both arrival sequences A, A' . It suffices to prove that on both arrival sequences, mf sends the same packet in the first time step and on both cases the buffer is equal at the end of the first time step. Since mf does not drop an expensive packet at the first time we get that it is enough to prove that both buffers have the same cheap packet with the same mark.

Let n_1 be the number of cheap packets that arrive before packet p_2 and are marked by expensive packets that arrive after packet p_2 i.e. $n_1 = |\{p \in A: v(p) = 1, i^A(p) < i^A(p_2) \leq n^A(p)\}|$. We denote by n_x the number of expensive packets that arrive after p_2 . Note that $n_x = p_3 - p_2$.

Now let us look at the arrival sequence A' . In order to finish the proof we analyze three cases.

First assume that $p \in D_{\text{mf}}^A(1) \cap A'(1)$ and $n^A(p) < i^A(p_2)$. In this case p is marked by the same expensive packet i.e., $n^{A'}(p) = n^A(p)$, since p_2 arrives after the time p is marked and therefore the packet p_2 has no influence on the mark of p .

We turn to the second case, assume that $p \in D_{\text{mf}}^A(1) \cap A'(1)$ and $i^A(p_2) \leq n^A(p)$. Clearly $n^A(p) \leq n^A(p_1)$. Using the fact that p_2 is the last cheap packet dropped at the first step, we get that the number of expensive packets that arrived before p_3 is enough to reject $n_1 + 1$ cheap packets i.e., $n_x r \geq n_1 + 1$. Since the number of cheap packets that were not marked by expensive packets arriving before p_2 in A' is n_1 , we get that all cheap packets that arrive before p_2 in A' are marked by expensive packets arriving before p_3 . From the fact that the packet $n^A(p_1) > i^A(p_2)$ we get that p_1 is the last cheap packet that is marked by an expensive packet arriving before p_3 .

From these two cases we get that $D_{\text{mf}}^{A'}(1) = D_{\text{mf}}^A(1) \cup \{p_2\}$.

Finally, we have to prove that every cheap packet p with $i^A(p_3) \leq i^A(p)$ has the same mark in both arrival sequences A, A' . This follows from the fact that all cheap packets that arrive before p_3 are marked by expensive packets that arrive before p_3 . Therefore, expensive packets that arrive after p_3 can only mark cheap packets that arrive after p_3 in both arrival sequences. Since we did not change the packets that arrive after p_2 we get that $n^{A'}(p) = n^A(p)$ and this completes the proof. \square

Lemma 4.10. *Let A be an arrival sequence with no splitting points such that $v(S_{\text{opt}}(1)) = 1$, $v(S_{\text{mf}}(1)) = \alpha$, and opt does not drop any packet during steps $1, \dots, t_z$. Then $\text{cr}(A) \leq (r + \alpha)/\alpha$.*

Proof. Let n_x and n_1 be the number of expensive and cheap packets in A , respectively, and let n'_1 be the number of cheap packets mf sends. First, note that the duration of A is t_z : since opt accepts all packets, $|Q_{\text{opt}}(t)| \geq |Q_{\text{mf}}(t)|$ for all $1 \leq t \leq t_z$. This means, in particular, that $Q_{\text{mf}}(t_z) = \emptyset$, and hence t_z is a splitting point unless it is the last step of A . Next, we argue that mf does not discard any expensive packet in steps $1, \dots, t_z$: since opt does not drop any packet, $\exp(Q_{\text{mf}}(t)) \leq \exp(Q_{\text{opt}}(t))$ for all $1 \leq t \leq t_z$, and hence, mf does not have any over-

flow and therefore mf never discards an expensive packet. It follows that $\text{cr}(A) = (n_1 + n_\alpha \alpha) / (n'_1 + n_\alpha \alpha)$. In addition, note that from the definition of mf it follows that $n_1 \leq n'_1 + rn_\alpha$. Hence we get that $\text{cr}(A) \leq (n'_1 + rn_\alpha + n_\alpha \alpha) / (n'_1 + n_\alpha \alpha)$. This last expression is maximized when $n'_1 = 0$, and thus we conclude

$$\text{cr}(A) \leq \frac{(r + \alpha)n_\alpha}{n_\alpha \alpha} \leq \frac{r + \alpha}{\alpha}. \quad \square$$

4.3. Putting the pieces together

We now arrive at the main result of this paper: the competitive ratio of algorithm mf. We state the bound with α and r as parameters.

Lemma 4.11. *For all arrival sequences A ,*

$$\frac{v_{\text{opt}}(A)}{v_{\text{mf}^r}(A)} \leq \max \left(\frac{(2+r)\alpha}{1+\alpha(1+r)}, \frac{r+\alpha}{\alpha} \right).$$

Proof. The proof is by induction on $|A|$. The base case is the empty sequence A_0 , for which we define $\text{cr}(A_0) = 1$. For the induction step, assume that the theorem holds for all arrival sequences with less than $|A|$ packets. We proceed by case analysis.

If A is splittable, then by Lemma 4.2 there exists an arrival sequence A' such that $|A'| < |A|$ and $\text{cr}(A') \geq \text{cr}(A)$, and we are done by induction. If $v(S_{\text{mf}}^A(1)) = v(S_{\text{opt}}^A(1))$, then by Lemma 4.3 there exists an arrival sequence A' such that $|A'| < |A|$ and $\text{cr}(A') \geq \text{cr}(A)$, and we are done by induction.

It remains to deal with the case that A is un-splittable and $v(S_{\text{mf}}^A(1)) \neq v(S_{\text{opt}}^A(1))$. We distinguish between two cases here. If $v(S_{\text{opt}}(1)) = \alpha$ and $v(S_{\text{mf}}(1)) = 1$, we have two sub-cases:

- If $\text{exp}(Q_{\text{mf}}(t)) = B$ only when $\text{exp}(Q_{\text{opt}}(t)) = B$, then by Lemma 4.7 we have that $\text{cr}(A) \leq (2+r)\alpha / (1+\alpha(1+r))$ and we are done.
- Otherwise, by Lemma 4.6, there exists an arrival sequence A' such that $|A'| < |A|$ and $\text{cr}(A') \geq \text{cr}(A)$, and we are done by induction.

If $v(S_{\text{opt}}(1)) = 1$ and $v(S_{\text{mf}}(1)) = \alpha$, we again have two sub-cases to consider.

- If opt does not drop any packet in A , we have that $\text{cr}(A) \leq (r + \alpha) / \alpha$ by Lemma 4.10 and we are done.
- Otherwise, by Lemma 4.9, there exists an arrival sequence A' such that $|A'| < |A|$ and $\text{cr}(A') \geq \text{cr}(A)$, and we are done by induction. \square

4.3.1. Tuning the parameters

To get the optimal algorithm, we write

$$\frac{\alpha(r+2)}{\alpha(r+1)+1} = \frac{\alpha+r}{\alpha}. \quad (3)$$

To solve Eq. (3), we first find what is the best r for a given α , and then find the worst-case α . It turns out that the optimal r for a given value of α is

$$\begin{aligned} r(\alpha) &= \frac{-1 - \alpha + \sqrt{4(\alpha - 1)\alpha^2 + (1 + \alpha)^2}}{2\alpha} \\ &= \sqrt{\alpha} + \frac{1}{2} + O\left(\frac{1}{\sqrt{\alpha}}\right). \end{aligned}$$

For this choice of r , we get that the competitive ratio is

$$\begin{aligned} \text{cr}_{\text{mf}^r} &= \frac{-1 - \alpha + 2\alpha^2 + \sqrt{1 + 2\alpha - 3\alpha^2 + 4\alpha^3}}{2\alpha^2} \\ &= 1 + \frac{1}{\sqrt{\alpha}} + O\left(\frac{1}{\alpha}\right). \end{aligned}$$

Solving numerically (see Fig. 5), we find that the worst case occurs when $\alpha \approx 3.751$, and then the competitive ratio is approximately 1.304.

4.3.2. Dealing with non-integral r

If r is non-integral, the algorithm is modified as follows. Each packet may have a “fractional mark”, so that each expensive packet fills the marks going top-down. Only a packet that is fully marked is preempted. The $n(p)$ is the index of the

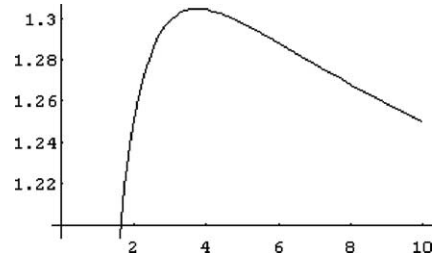


Fig. 5. Behavior of the competitive ratio of mf as a function of α .

packet that completed the fractional mark of p to 1. If the packet at the bottom of the buffer is not fully marked, then no preemption occurs. The analysis of the algorithm carries over unchanged, except for some minor adjustments.

5. Conclusions

We have presented a buffer management algorithm for the DiffServ model, where only two packet values are possible. The obvious open question is whether the upper bound can be improved to meet the lower bound (we believe that the lower bound is the best possible).

Another important open problem is finding an algorithm for the general model, where packet values may be any number in $[1, \infty]$. In fact, it is not clear even how to generalize our algorithm to more than two values.

Lastly, we remark that from the implementation point of view, head-drop and tail-drop are extremely more efficient than push-out queues. We believe that our algorithm can be extended to the head-drop model.

Acknowledgement

We thank Alex Kesselman for detecting an important error in one of the earlier drafts of this paper.

References

- [1] W. Aiello, Y. Mansour, S. Rajagopalan, A. Rosen, Competitive queue policies for differentiated services, in: Proceedings of the IEEE INFOCOM, 2000.
- [2] D. Black, S. Blake, M. Carlson, E. Davies, Z. Wang, W. Weiss, An architecture for differentiated services, Internet RFC 2475, December 1998.
- [3] A. Borodin, R. El-Yaniv, *Online Computation and Competitive Analysis*, Cambridge University Press, Cambridge, MA, 1998.
- [4] D. Clark, J. Wroclawski, An approach to service allocation in the Internet, Internet draft, 1997. Available from diffserv.lcs.mit.edu.
- [5] D.D. Clark, A model for cost allocation and pricing in the Internet, in: L. McKnight, J. Bailey (Eds.), *MIT Workshop on Internet Economics*, MIT Press, Cambridge, MA, 1997.
- [6] R. Cruz, A calculus for network delay. Part I. Network elements in isolation, *IEEE Trans. Info. Theory* 37 (1) (1991) 114–131.
- [7] S. Floyd, V. Jacobson, Random early detection gateways for congestion avoidance, *IEEE/ACM Trans. Networking* 1 (4) (1993) 397–413.
- [8] F. Kelly, A. Maulloo, D. Tan, Rate control in communication networks: shadow prices, proportional fairness and stability, *J. Operat. Res. Soc.* 49 (1998) 237–252.
- [9] A. Kesselman, Z. Lotker, Y. Mansour, B. Patt-Shamir, B. Schieber, M. Sviridenko, Buffer overflow management in QoS switches, in: *Proceedings of the 33rd ACM STOC*, July 2001, pp. 520–529.
- [10] A. Kesselman, Y. Mansour, Loss-bounded analysis for differentiated services, in: *Proceedings of the 12th Annual ACM-SIAM Symposium on Discrete Algorithms*, 2001, pp. 591–600.
- [11] M.A. Labrador, S. Banerjee, Packet dropping policies for ATM and IP networks, *IEEE Commun. Surveys* 2 (3) (1999) 2–14.
- [12] W.E. Leland, M.S. Taqqu, W. Willinger, D.V. Wilson, On the self-similar nature of ethernet traffic (extended version), *IEEE/ACM Trans. Networking* 2 (1) (1994) 1–15.
- [13] Z. Lotker, Personal Communication, October 2000.
- [14] J.K. MacKie-Mason, H.R. Varian, Pricing the Internet, in: B. Kahin, J. Keller (Eds.), *Public Access to the Internet*, MIT Press, Cambridge, MA, 1995.
- [15] Y. Mansour, B. Patt-Shamir, O. Lapid, Optimal smoothing schedules for real-time streams, in: *Proceedings of the 19th Annual ACM Symposium on Principles of Distributed Computing*, Portland, OR, July 2000, pp. 21–30.
- [16] M. May, J.-C. Bolot, A. Jean-Marie, C. Diot, Simple performance models of differentiated services for the Internet, in: *Proceedings of the IEEE INFOCOM*, 1998.
- [17] D.D. Sleator, R.E. Tarjan, Amortized efficiency of list update and paging rules, *Commun. ACM* 28 (2) (1985) 202–208.
- [18] M. Sviridenko, A lower bound for on-line algorithms in the FIFO model, Unpublished manuscript, April 2001.
- [19] The ATM Forum Technical Committee, Traffic management specification version 4.0, April 1996. Available from www.atmforum.com.
- [20] A. Veres, M. Boda, The chaotic nature of TCP congestion control, in: *Proceedings of the IEEE INFOCOM*, 2000, pp. 1715–1723.

Zvi Lotker holds a B.Sc. in Mathematics and Computer Science, a B.Sc. in Industrial Engineering (both from Ben Gurion University), and an M.Sc. in Mathematics from Tel Aviv University. He is currently a Ph.D. candidate in the Department of Electrical Engineering in Tel Aviv University. His thesis topic is network algorithms.

Boaz Patt-Shamir has received his B.Sc. in Mathematics and Computer Science from Tel Aviv University, his M.Sc. in Computer Science from Weizmann Institute, and his Ph.D. in Computer Science from MIT. Following that he was on the faculty of Northeastern University. Since 1997 he is with the Department of Electrical Engineering in Tel Aviv University, where he directs the laboratory for computer networks and multimedia. His main research interests include distributed systems and network algorithms.