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

ELSEVIER

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.

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).

1389-1286/03/$ - see front matter
doi:10.1016/S1389-1286(03)00197-X

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'e 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.