Optimal Buffer Control During Congestion in an ATM Network Node

Leandros Tassiulas, *Member, IEEE,* Yao Chung Hung, and Shivendra S. Panwar, *Member, IEEE*

***Abstract*—In this paper we study the problem of optimal buffer space priority control in an ATM network node. The buffer of a transmission link is shared among the cells of several traffic classes waiting for transmission through the link. When the number of cells to be stored in the buffer exceeds the available buffer space, certain cells have to be dropped. Different traffic classes have different sensitivities to cell losses. By appropriately selecting the classes of cells which are dropped or blocked in case of overflow, we can have the more sensitive classes suffer smaller cell losses. Depending on the control that we have on the system, three classes of policies are distinguished. In each one, policies that schedule the buffer allocation in some optimal manner are identified.**

I. Introduction

ONE of the main problems arising in the area of high speed communication networks is the design of control algorithms for the efficient sharing of the buffer space in an ATM node. Cells of different traffic types arrive at the node and are stored in a buffer until their transmission. Cells of different types may be generated by a leaky bucket policing function which marks excessive traffic cells at the source network interface or by an encoding scheme which creates cells with different priorities [2]. When a cell finds the buffer full upon arrival, it may be discarded before admission into the system. The cell loss due to buffer overflow incurs a degradation in the overall system performance which is highly dependent on the type of the discarded cells. Certain traffic types are more sensitive to potential cell losses than others. We can reduce the probability of discarding a loss-sensitive cell due to buffer overflow if we block the admission of less loss sensitive cells. We may also consider expelling less loss sensitive cells from the buffer. In this paper we study how we can do this in an optimal manner.

We consider a single outgoing link and the corresponding dedicated buffer in a network node. The system is modeled by a single server queue (Fig. 1). The queue has a buffer that can store B cells; this is called the *main buffer* in the following. Time is slotted and the transmission of a cell takes one slot. During one slot at most B_T cells may arrive to the system and they are placed in the *temporary buffer* which has length B_T. These cells may belong to different traffic types. This assumption is consistent with the structure of knockout-type ATM switches [14] or a switch with output queueing. At the end of each slot the cells from the temporary buffer are either placed in the main buffer or dropped from the system. Depending on the available control we have over the dropping of cells from the temporary or the main buffer and over the placement of the cells in the main buffer, we will distinguish three classes of policies. In all the policies considered it is assumed that the cells which enter the main buffer in every slot should join the end of the queue and rearrangement of cells is not allowed. Hence, the FIFO discipline is preserved and the cells are delivered in order. This property is essential in virtual circuit connections.

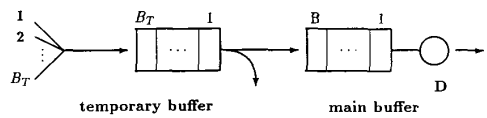

Fig. 1. The system model.

The first class is that of *discarding policies.* A discarding policy cannot modify the state of the cells which are already in the main buffer. It can only control the admission of the cells from the temporary buffer, by blocking some if necessary, and the placement of the admitted cells in the main buffer. We show that the optimal discarding policy is of "multithreshold type." That is, for each priority class there is a threshold, and if the number of cells in the main buffer exceeds that threshold, the cells of that class are blocked from admission. The policy is optimal in the sense that it minimizes the long run average blocking cost where a cost is associated with each cell that reflects the loss sensitivity of its class.

The second class of policies considered are the *pushout policies.* A pushout policy is allowed to expel cells from the main buffer in order to make space for cells in the temporary buffer which cannot enter the main buffer because it is full. A cell from the temporary buffer cannot be blocked from admission to the main buffer if there is space in the main buffer. We obtain the optimal pushout policy, which we call the *squeeze-out policy*, in a system with two priority classes. That policy places the cells in the main buffer, high priority first. If the buffer is full and there are cells in the temporary buffer, then the low-priority cells are pushed out of the main buffer starting from those closest to the head of the queue. Notice that low-priority cells are dropped to make space for other low-

Manuscript received January 17, 1993; revised August 3, 1993 and May 2, 1994; approved by IEEE/ACM TRANSACTIONS ON NETWORKING Editor I. Cidon. This work was supported by the Center for Advanced Technology in Telecommunications, Polytechnic University and by the NSF under Grants NCR-8909719, NCR-9115864, and NCR-9211417. This paper was presented in part at the Proceedings of the 1992 Conference on Information Sciences and Systems, Princeton, NJ, March 18-20, 1992.

The authors are with the Department of Electrical Engineering and Center for Advanced Technology in Telecommunications, Polytechnic University, Brooklyn, NY 11201 USA.

IEEE Log Number 9404837.

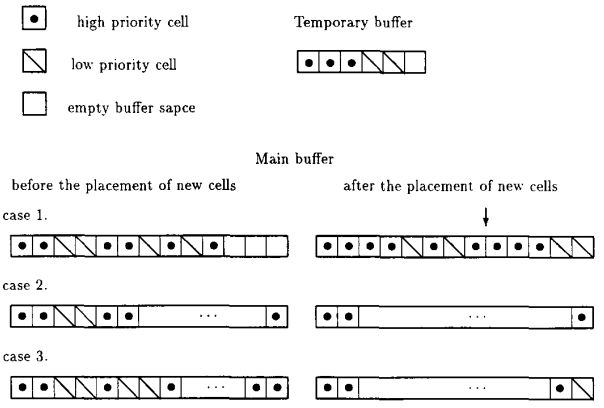

Fig. 2. Buffer configurations before and after the placements of new cells.

priority cells that are appended to the end of the queue. The squeeze-out policy minimizes the blocking probability of the high-priority (loss sensitive) class among all pushout policies.

The third class of policies considered are the *expelling policies*. Expelling policies are allowed to expel cells from the main buffer or block cells in the temporary buffer from admission into the main buffer irrespective of the system state. Properties of the optimal expelling policy are obtained that narrow down the set of candidate policies considerably in a system with two classes. More specifically, we show that, like in the case of the squeeze-out policy, the cells are placed in the main buffer high priority first and low-priority cells are pushed out, starting from the head of the queue, if there is no space. In addition to that, the optimal expelling policy may drop low-priority cells even if the main buffer is not full, but only if the low-priority cell(s) is (are) at the head of the queue.

Clearly, an expelling policy has more control over the system than discarding and pushout policies. In other words, the class of expelling policies contains the discarding and pushout policies as subclasses. Policies of different classes have different degrees of implementation difficulty. For one approach that allows for the implementation of some of the policies considered in this paper, see [3].

The problem of sharing the buffer space among several competing traffic streams has attracted considerable attention in the past. Several strategies for buffer sharing, called space priority access methods, have been proposed and analyzed. Petr and Frost in [10] distinguish several classes of buffer sharing policies based on the time instances at which control actions can be taken and on the groups of cells that can be discarded. The three classes of policies studied here fall within that framework. Discarding type policies have been studied by Petr and Frost in [9], [11]. In [9] the problem of minimizing the average discarding cost has been considered in a system with an arbitrary number of priority classes and one buffer space. In [11] the problem of maximizing the offered load over all multithreshold type policies under constraints on the losses of each class is considered. Here we determine the optimum discarding policy for systems with buffers of arbitrary length.

The pushout scheme is another buffer sharing strategy that has been studied extensively in the past. An important component of a pushout strategy is to decide which cell to pushout of the buffer in order to make space for an incoming cell. Kroner and Kroner *et al.* have analyzed the performance of several buffer sharing schemes, in [6], [7] including a pushout policy which expells low-priority cells starting from those closest to the tail of the queue, i.e., the youngest low-priority cells. They obtained the cell loss probabilities under different buffer sharing schemes for a two class $M/G/1/N$ system. In our work we identify two important properties of the optimal pushout policy. It is better to push out the oldest low-priority cell from the buffer and it is better to push out a low-priority cell from the buffer in order to make space for another cell, irrespective of its priority. These two properties uniquely characterize the optimal pushout policy, called the squeeze-out policy, as we show in Section III–A. The class of expelling policies has been identified in [10] but they haven't been analyzed. Lippman [8] showed that the optimum discarding policy for an $M/M/c/K$ queue (with no temporary buffer) is of multi-threshold type. Policies to meet both cell delay and loss requirements were considered in [1], [5].

The paper is organized as follows. In Section II the discarding policies are analyzed. The pushout and the expelling classes of policies are analyzed in Section III–A and –B, respectively. In Section IV we discuss some of the implications of our results. In Section V numerical results are reported. In Section VI we discuss some extensions to our work and open problems.

II. Discarding Policies

The cells are classified into L priority classes. The high-priority classes are more sensitive to cell losses. Without loss of generality we assume that the priority of class l is higher than the priority of class $l+1$. The priority of a class is reflected by the cost that is incurred by the blocking of a cell of that class. As we mentioned earlier, at most B_T cells of all classes arrive into the system during every slot and they reside in the temporary buffer. By the end of each slot a decision is taken regarding which cells will be admitted in the system and where they are going to be placed in the buffer. The rest of the cells are discarded. We denote by $X_i^M(t)$ the class of the cell residing at the main buffer position i, $i = 1,..,B$ by the end of slot t; $X_i^M(t) = 0$ if position i is empty at this time. We denote by $X_i^T(t)$ the class of the cell residing at position i of the temporary buffer $i = 1,..,B_T$; $X_i^T(t) = 0$ if this position is empty at this time. The vectors $\mathbf{X}^M(t) = (X_i^M(t) : i = 1,..,B)$, $\mathbf{X}^T(t) = (X_i^T(t) : i = 1,..,B_T)$, represent the main and temporary buffer occupancies at the end of slot t. Without loss of generality we may assume that in the temporary buffer, the cells are stored in decreasing priority order and in contiguous buffer spaces; that is, for $X_i^T(t) > 0$, $i > 1$, we have $0 < X_{i-1}^T(t) \leq X_i^T(t)$. The temporary buffer at the end of slot t contains cells that arrived during slot t only. The ordering of the cells in the temporary buffer is assumed only for a less cumbersome discription of the optimal discarding policy, since

in this manner the control action can be specified by a single variable. The cells which are admitted can be placed in any order in the main buffer since, as it is shown later, the ordering of the cells in the main buffer is irrelevant to the performance of the optimal discarding policy. This is not the case for the pushout and expelling policies where the order of the cells in the main buffer is important. The implications of this property for these two classes of policies are discussed in Section III. We assume independent identically distributed arrivals from slot to slot. The vector $\mathbf{X}(t) = (\mathbf{X}^M(t), \mathbf{X}^T(t))$ is a natural state variable and we use the notation $\{\mathbf{X}(t), t \geq 0\}$ for the stochastic process that describes the evolution of the system. The state space of that process is $\mathcal{X} = \mathcal{X}^M \times \mathcal{X}^T$ where $\mathcal{X}^M = \{0, 1, .., L\}^B$ and $\mathcal{X}^T = \{0, 1, .., L\}^{B_T}$ are the spaces where the vectors $\mathbf{X}^M(t)$ and $\mathbf{X}^T(t)$ lie respectively.

All the cells in the temporary buffer, by the end of each slot t, are either admitted in the system and placed in the main buffer or rejected. We control the admission of cells in the main buffer. The control actions taken by the end of slot t are represented by the *admission* variables $A_i(t) \in \{0, 1, .., B\}$, $i = 1, .., B_T$ as follows. We have $A_i(t) = 0$ if either position i of the temporary buffer is empty or the cell stored in that position is blocked from admission into the system; we have $A_i(t) = j$ if the cell residing in position i of the temporary buffer is placed in position j of the buffer. The vector $\mathbf{A}(t) = (A_i(t) : i = 1, .., B_T)$ is called the admission vector at time t in the following. Let $\mathcal{A} = \{0, .., B\}^{B_T}$ be the space where it lies; this is called action space in the following. We assume that the cells of the temporary buffer which are admitted in the main buffer are placed in consecutive positions at the end of the existing queue. Let $S(\mathbf{x})$ be the set of all admission vectors which satisfy the above assumption when the system is in state $\mathbf{x}$.

At each slot t exactly one cell is transmitted. The cells in the main buffer are served in a FIFO manner. Given the state of the system at the end of slot t and the admission vector at that time, the main buffer occupancy vector by the end of slot $t + 1$ is specified deterministically. Let $D : \mathcal{X} \times \mathcal{A} \to \mathcal{X}^M$ be a function such that $\mathbf{X}^M(t + 1) = D(\mathbf{X}(t), \mathbf{A}(t))$. The state of the temporary buffer at the end of slot $t + 1$ is determined completely from the arrivals during that slot. Under the assumption of i.i.d. arrivals the evolution of the system is Markovian. Given the state of the system at time t and the admission vector at that time, the probability distribution of the state at $t + 1$ is completely determined by the function D and the probability distribution induced by the arrivals on $\mathcal{X}^T$. Let $p_{\mathbf{y}}$ be the probability that the temporary buffer has the configuration $\mathbf{y}$, $\mathbf{y} \in \mathcal{X}^T$ at the end of slot $t + 1$. The transition probability $P_{\mathbf{x}\mathbf{x}'}(\mathbf{a}) = Pr(\mathbf{X}(t + 1) = \mathbf{x}' | \mathbf{X}(t) = \mathbf{x}, \mathbf{A}(t) = \mathbf{a})$ is given by

$$P_{\mathbf{x}\mathbf{x}'}(\mathbf{a}) = \begin{cases} 0, & \text{if } \mathbf{x}'^M \neq D(\mathbf{x}, \mathbf{a}) \\ p_{\mathbf{y}}, & \text{if } \mathbf{x}'^M = D(\mathbf{x}, \mathbf{a}), \mathbf{x}'^T = \mathbf{y}, \mathbf{y} \in \mathcal{X}^T. \end{cases} \tag{1}$$

An *admission policy* is any rule for selecting the admission variables at every time $t \geq 0$. This decision is made on the basis of the past system states $\{\mathbf{X}(s), t \geq s \geq 0\}$ and past decisions. Let G be the class of all admission policies such that the admission vector $\mathbf{A}(t)$ belongs to the set $S(\mathbf{X}(t))$ at all t.

When a cell of class l is dropped from the system then a cost c_l is incurred. We assume that the classes are indexed in decreasing priority, that is $c_l > c_{l+1}$, $l = 1, .., L - 1$. By convention we set $c_0 = 0$. The total cost incurred when the system is in state $\mathbf{x}$ and the admission actions that correspond to vector $\mathbf{a} \in S(\mathbf{x})$ are taken is

$$c(\mathbf{x}, \mathbf{a}) \stackrel{\text{def}}{=} \sum_{i=1}^{B_T} 1\{a_i = 0\} c_{x_i^T}, \mathbf{x} \in \mathcal{X}, \mathbf{a} \in S(\mathbf{x}). \tag{2}$$

where $1\{a_i = 0\} = 1$ if $a_i = 0$, 0 otherwise.

The blocking cost incurred at time t is $C(t) = c(\mathbf{X}(t), \mathbf{A}(t))$. Our objective is to minimize the average blocking cost. The long run average cost associated with a policy $g \in G$ is defined by

$$J_g(\mathbf{x}) \stackrel{\text{def}}{=} \limsup_{T \to \infty} E_{\mathbf{x}}^g \frac{1}{T} [\sum_{t=0}^{T-1} C(t)], \mathbf{x} \in \mathcal{X} \tag{3}$$

where $E_{\mathbf{x}}^g[\cdot]$ denotes the expectation with respect to the probability measure induced by the policy g on the state process starting in state $\mathbf{x}$. An admission policy g_D is said to be average cost optimal discarding policy if it minimizes (3) within G, i.e., if

$$J_{g_D}(\mathbf{x}) \leq J_g(\mathbf{x}), \mathbf{x} \in \mathcal{X}$$

for any other policy $g \in G$. Under our assumptions about the arrival statistics, the optimization problem associated with (3) falls within the family of discrete time Markov Decision Processes (MDP's). Since the state space is finite , it is well known that an optimal policy exists and it can be taken in the class of Markov stationary policies [12]), that is policies for which the control actions are time-invariant functions of the state only. A stationary policy g is identified by the functions $g_i : \mathcal{X} \to \{0, 1, .., B\}$, $i = 1, .., B_T$. When the system is controlled under policy g, at every time t we have $A_i(t) = g_i(\mathbf{X}(t))$ and in vector form $\mathbf{A}(t) = g(\mathbf{X}(t))$. In order to study the optimization problem associated with the long run average cost (3) we need to consider first the optimization problem associated with the β-discounted cost defined next.

The β-discounted cost $(0 < \beta < 1)$ associated with a policy $g \in G$ is defined by

$$V_g^\beta(\mathbf{x}) \stackrel{\text{def}}{=} E_{\mathbf{x}}^g [\sum_{t=0}^{\infty} \beta^t C(t)], \mathbf{x} \in \mathcal{X} \tag{4}$$

where $E_{\mathbf{x}}^g[\cdot]$ has the same meaning as in (3). An admission policy g_D^β is said to be β-optimal discarding policy if it minimizes (4) within G, i.e., if

$$V_{g_D}^\beta(\mathbf{x}) \leq V_g^\beta(\mathbf{x}), \mathbf{x} \in \mathcal{X}$$

for any other policy $g \in G$. It is well known [12] that a β-optimal policy exists and it can be taken within the

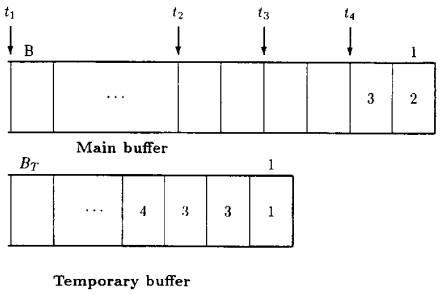

Fig. 3. An example of optimal discarding policy with four classes.

Markov stationary policies. The β-optimal cost associated with discarding policies is by definition

$$V^{\beta}(\mathbf{x}) = \inf_{g \in G} V_g^{\beta}(\mathbf{x}), \mathbf{x} \in \mathcal{X}.$$

It is also well known that since the Markov decision process under consideration has finite state space, the β-optimal cost is achieved for some policy and it satisfies the dynamic programming equation

$$V^{\beta}(\mathbf{x}) = \min_{\mathbf{a} \in S(\mathbf{x})} \{c(\mathbf{x}, \mathbf{a}) + \beta \sum_{\mathbf{x}' \in \mathcal{X}} P_{\mathbf{x}\mathbf{x}'}(\mathbf{a}) V^{\beta}(\mathbf{x}')\}. \quad (5)$$

In view of (1), (5) can be written as

$$V^{\beta}(\mathbf{x}) = \min_{\mathbf{a} \in S(\mathbf{x})} \{c(\mathbf{x}, \mathbf{a}) + \beta \sum_{\mathbf{y} \in \mathcal{X}^T} p_{\mathbf{y}} V^{\beta}(D(\mathbf{x}, \mathbf{a}), \mathbf{y})\}.$$

The necessary and sufficient condition for a stationary policy g^{β} to be a β-optimal discarding policy is

$$g^{\beta}(\mathbf{x}) = \arg \min_{\mathbf{a} \in S(\mathbf{x})} \{c(\mathbf{x}, \mathbf{a}) + \beta \sum_{\mathbf{y} \in \mathcal{X}^T} p_{\mathbf{y}} V^{\beta}(D(\mathbf{x}, \mathbf{a}), \mathbf{y})\}.$$

A. Optimal Cell Discarding Policy

In this section we study the problem of obtaining the optimal discarding policy. We show that the optimal policy makes the admission decisions based only on the length of the main buffer and not on the class of packets in it. Furthermore, it is of threshold type with one threshold for each class of cells. According to the optimal policy (Fig. 3), cells are admitted in the main buffer high-priority cell first. If the queue length exceeds the threshold t_i and the next cell to enter the main buffer is of class t_i, then the rest of the cells are discarded. We characterize the optimal policy for the β-discounted problem. Then by standard arguments [12] the same characterization is obtained for the average cost optimal policy.

1) State Space Reduction: In this section we show that the decisions of the optimal policy depend only on the number of cells in the main buffer and on the state of the temporary buffer. Based on that, we get a reduction of the state space. Let $l(\mathbf{x})$ be the number of cells in the main buffer when the system is in state $\mathbf{x}$. In the next lemma, we show that the β-optimal cost has the same value for any two states that correspond to the same temporary buffer state and the same number of messages in the main buffer. The proof of the lemma is in the appendix.

Lemma 2.1: For any two states $\mathbf{x}_0 = (\mathbf{x}_0^M, \mathbf{x}_0^T)$, $\mathbf{x}_1 = (\mathbf{x}_1^M, \mathbf{x}_1^T)$ in $\mathcal{X}$ such that $l(\mathbf{x}_0) = l(\mathbf{x}_1)$ and $\mathbf{x}_0^T = \mathbf{x}_1^T$, the value function for the β-discounted problem satisfies

$$V^{\beta}(\mathbf{x}_0) = V^{\beta}(\mathbf{x}_1).$$

From the above lemma and the dynamic programming equation for the β-discounted problem the next theorem follows readily.

Theorem 2.1: For any two states $\mathbf{x}_0 = (\mathbf{x}_0^M, \mathbf{x}_0^T)$, $\mathbf{x}_1 = (\mathbf{x}_1^M, \mathbf{x}_1^T)$ in $\mathcal{X}$ such that $l(\mathbf{x}_0) = l(\mathbf{x}_1)$ and $\mathbf{x}_0^T = \mathbf{x}_1^T$, a β-optimal policy g^{β} satisfies

$$g^{\beta}(\mathbf{x}_0) = g^{\beta}(\mathbf{x}_1).$$

From the above results it is clear that when we consider cell discarding policies we may consider the Markov decision process defined on the state space $\hat{\mathcal{X}} = \{0, 1, .., B\} \times \mathcal{X}^T$ which will be denoted by $\mathcal{X}$ in the rest of this section.

From now on the state of the process is $\mathbf{x} = (i, \mathbf{x}^T)$ where $i \in \{0, 1, .., B\}$ and $\mathbf{x}^T \in \mathcal{X}^T$. The action space associated with a state $\mathbf{x} = (i, \mathbf{x}^T)$ is equal to the common action space $S(\mathbf{x}')$ of all states $\mathbf{x}' \in \mathcal{X}^M \times \mathcal{X}^T$ such that $l(\mathbf{x}') = i$. An immediate implication of the above reduction of the state space is that the placement of the admitted cell in the main buffer is irrelevant as far as the optimal control problem is concerned. Therefore, the action vector should indicate for every cell in the temporary buffer whether it is admitted or not and can be taken to be binary. Indeed, the action vectors will be considered to be binary in the following. The transition operator $D(\cdot, \cdot)$ indicates the length of the main buffer in the new state space and it takes values in Z^+. If the action vector $\mathbf{a} \in S(\mathbf{x})$ has n nonzero elements and $\mathbf{x} = (i, \mathbf{x}^T)$ then the transition operator is $D(\mathbf{x}, \mathbf{a}) = i + n - 1$ if $i \neq 0$, and $D(\mathbf{x}, \mathbf{a}) = i + n$ if $i = 0$.

2) Reduction of the Action Space: The reduction of the state space that was obtained in the previous section implies a reduction in the action space from B_T-dimensional to one dimensional. More specifically we will show that the optimal action is to accept the first n cells in the temporary buffer for some value of n. The placement in the main buffer of the accepted cells is irrelevant. Consider the activation vectors

$$\mathbf{a}_n = (a_j : a_j = 1,\ 1 \leq j \leq n,\ a_j = 0,\ n < j \leq B_T).$$

The following theorem is proved in the appendix.

Theorem 2.2: The minimum in the right-hand side of the dynamic programming equation (5) is achieved at one of the action vectors $\mathbf{a}_n$, $n = 0, \cdots, B_T$ and the dynamic programming equation can be written as

$$V^{\beta}(\mathbf{x}) = \min_{n=0,\cdots,B_T} \{ \sum_{j=n+1}^{B_T} c_{\mathbf{x}_j^T} + \beta \sum_{y \in \mathcal{X}^T} p_y V^{\beta}(1\{i > 0\}(i + n - 1) + 1\{i = 0\}(i + n), y)\}. \quad (6)$$

The β-optimal discarding policy $g^\beta(\mathbf{x})$ can be considered to take values in $\{0,1,\cdots,B_T\}$ and it is

$$g^\beta(\mathbf{x}) = \arg \min_{n=0,\cdots,B_T} \{ \sum_{j=n+1}^{B_T} c_{\mathbf{x}_j^T} + \beta \sum_{y \in \mathcal{X}^T} p_y V^\beta(1\{i>0\}(i+n-1) + 1\{i=0\}(i+n), y)\}. \quad (7)$$

B. The Optimal Policy

The characterization of the optimal policy is given after the next lemma. Two properties of the cost function are stated. First that it is increasing with the queue size of the main buffer, or in other words that more cells are discarded in the future if there are more cells in the main buffer initially. Second, that the cost function is convex, that is the increase of the number of discarded cells in the future, as we go from initial main buffer queue size i to size $i+1$, increases with i. Roughly speaking, the rate of increase of the cost function with respect to i is increasing as well.

Lemma 2.3: The value function associated with the β-discounted problem for discarding policies is nondecreasing and convex in i

$$V^\beta((i,\mathbf{x}^T)) \le V^\beta((i+1,\mathbf{x}^T)),\ i=0,\cdots,B-1$$

$$V^\beta((i,\mathbf{x}^T)) - V^\beta((i-1,\mathbf{x}^T)) \le V^\beta((i+1,\mathbf{x}^T)) - V^\beta((i,\mathbf{x}^T)), i=1,..,B-1. \quad (8)$$

The proof of the lemma is cumbersome and is omitted here. It can be found in the technical report version of the paper [13]. A consequence of the convexity of the cost function is that the optimal policy will be of threshold type, as is stated rigorously in Theorem 2.3. This can be argued intuitively as follows. A cell of class j is accepted in the main buffer if the increase in the cost due to future discarded cells because of the admission of the current cell is smaller than the cost c_j that will be incurred by the discarded cell. Now the convexity of the cost function implies that the increase in the discarding cost is increasing with the main buffer queue size. Therefore if a cell of class j is discarded when the main buffer queue is equal to i, it will be discarded for larger main buffer lengths as well. The proof of the following theorem is in the appendix

Theorem 2.3: There exists a β-optimal discarding policy $g^\beta(i,\mathbf{y})$ of the following form. There are thresholds $t_1 \ge t_2 \ge .. \ge t_L$ defined by

$$t_j = \arg \min_{i\in\{1,\cdots,B\}} \{c_j + \beta \sum_{\mathbf{y}\in\mathcal{X}^T} p_y(V^\beta(i-2,\mathbf{y}) - V^\beta(i-1,\mathbf{y})) \ge 0.\} \quad (9)$$

In the case of ties, t_j is the maximum i which achieves the minimum above. A cell of class j in position k of the temporary buffer is accepted if and only if

$$t_j \ge i+k$$

where i is the length of the main buffer. That is

$$g^\beta(i,\mathbf{x}^T) = \max\{k : i+k \le t_{x_k^T}\}.$$

C. Average Cost Optimal Policy

Using standard techniques from the theory of Dynamic Programming we can characterize the optimal policy for the average cost case. The following theorem can be proved easily based on the results we obtained for the β-discounted problem and some results that relate the β-discounted and the average cost problem in [12]. The intuitive explanation of the thresholds is the same as for the β-discounted optimal policy.

Theorem 2.4: There exists an average cost optimal discarding policy g_D of the following form. Consider the thresholds $t_1 \ge t_2 \ge \cdots \ge t_L$ defined by

$$t_j = \arg \min_{i\in\{1,\cdots,B\}} \{c_j + \sum_{\mathbf{x}\in\mathcal{X}^T} p_y(h(i-2,\mathbf{x}) - h(i-1,\mathbf{x})) \ge 0\}$$

where $h(i,\mathbf{x}) = \lim_{n\to\infty}(V^{\beta_n}(i,\mathbf{x}) - V^{\beta_n}(0,0))$ for some sequence $\beta_n \to 1$ and in the case of ties, t_j is the maximum i which achieves the minimum above. A cell of class j in position k of the temporary buffer is accepted if and only if

$$t_j \ge i+k$$

where i is the length of the main buffer. That is

$$g_D(i,\mathbf{x}) = \max\{k : i+k \le t_{\mathbf{x}_k^T}\}.$$

We outline the steps of the proof of the Theorem 2.4. It is shown first that $\lim_{n\to\infty}(V^{\beta_n}(i,\mathbf{x}) - V^{\beta_n}(0,0))$ exists, therefore $h(i,\mathbf{x})$ is well defined. Then it is shown that $h(i,\mathbf{x})$ is increasing and convex in i using Lemma 2.3. The proof of the theorem is concluded in the same manner as the proof of Theorem 2.3.

III. Cell Expelling Policies

In the following two sections we study the pushout and the expelling policies. The main difference between these policies and the discarding policies is that cells in the main buffer can be dropped (expelled) from the main buffer under the pushout and expelling policies. The class of pushout policies contains all policies for which a cell is dropped only if the buffer is full. Any policy in the pushout class minimizes the total cell loss over all priority classes. Different pushout policies achieve a different apportioning of the individual losses of each priority class. In Section III–A we obtain the policy that minimizes the cell loss of the high-priority class in a system with two priority classes. This policy, that we call squeeze-out, achieves the minimization of cell losses in systems with arbitrary cell arrival processes.

The class of pushout policies is a subclass of the more general class of expelling policies. Policies of the latter class are allowed to expel or discard a cell at any time, even before the buffer is full. By dropping cells while the buffer is not full, the total cell loss is increased. However a smaller loss of the high-priority class may be achieved, than that achieved by the squeeze-out policy. In Section III–B, we identify the class of

policies G^{EO} with the following property. For any arbitrary expelling policy g, there is a policy $g' \in G^{EO}$, that has a total cell loss less than or equal to that of g and a high-priority cell loss less than or equal to that of g. Therefore for any practical purpose we can search for an expelling policy within G^{EO}. The class G^{EO} is considerably more restricted than the class of expelling policies. Unlike our study of discarding policies, the results in this section are restricted to systems with two priority classes only.

We keep the notation that we introduced in Section II in this section as well. Nevertheless we prefer to specify the class of policies we consider and the optimal policies in words rather than mathematically in this section, since the first description is precise enough and we don't need the mathematical description in the proof of the results.

A. Pushout Policies

The class of pushout policies G^p contains all policies which obey the following rules.

a) A cell can be expelled from the main buffer only if it is "pushed out" by another cell in the temporary buffer which cannot enter the main buffer because it is full.
b) A cell from the temporary buffer can be discarded only if the main buffer is full.

The following policy is optimal in G^p.

Squeeze-out Policy π^{po}:

1) Append the cells from the temporary buffer to the end of the main buffer, high-priority cells first.
2) If the main buffer is full, and there are cells in the temporary buffer, push out the low-priority cells starting from those closest to the head of the queue.
3) If all the low-priority cells in the main buffer are pushed out discard all remaining cells in the temporary buffer.

The policy π^{po} is optimal within G^p in a very strong sense. It minimizes at every slot t the number of high-priority cells lost. Furthermore, this holds for arbitrary arrival processes and not only for i.i.d. arrivals. Let $D^h(t), D^l(t)$ and $\tilde{D}^h(t), \tilde{D}^l(t)$ be the numbers of dropped cells by the end of slot t of the high and low-priority classes, respectively, under policy π^{po} and for an arbitrary policy $\tilde{\pi} \in G^p$. Then we have the following.

Theorem 3.1: When the system starts from the same initial state under policies $\pi^{po}, \tilde{\pi}$ and the arrivals are identical under the two policies we have

$$D^l(t) \geq \tilde{D}^l(t)$$
$$D^l(t) + D^h(t) = \tilde{D}^l(t) + \tilde{D}^h(t), \quad t = 1, 2, \cdots.$$

The intuition behind the optimality of the squeeze-out policy is the following. The total number of cells that go through the queue is the same for all policies of the pushout class. The policy that maximizes the throughput of the high-priority class, minimizes the throughput of the low-priority class. The squeeze-out policy does exactly this by maximizing the availability of low-priority cells for discarding from the main queue. The proof of the theorem relies on the following lemma and is included in the Appendix together with the proof of the lemma.

Lemma 3.1: For any policy $\tilde{\pi} \in G^p$ there exists a policy π_1 which acts identically to π^{po} at $t = 1$ and is appropriately defined at $t > 1$ such that if the system starts from the same initial state under both policies, and the arrival processes are identical, then we have

$$D^{1l}(t) \geq \tilde{D}^l(t)$$

$$D^{1l}(t) + D^{1h}(t) = \tilde{D}^l(t) + \tilde{D}^h(t), \quad t = 1, \cdots \tag{10}$$

where $D^{1l}(t), D^{1h}(t)$ are the numbers of dropped cells of low and high priority, respectively, by the end of slot t under policy π_1.

B. Expelling Policies

The class of expelling policies G^E has as members all policies that append the new cells from the temporary buffer to the end of the queue and do not rearrange the cells in the main buffer. An expelling policy is allowed to expel or block any cell in the main or temporary buffer respectively, irrespective of the state. Hence the only requirement an expelling policy should satisfy is to preserve the FIFO order. Other than that it can drop cells arbitrarily. Clearly the class of expelling policies is larger than the previous two.

We were able to obtain properties of the optimal policy that narrow down the class of policies that contains the optimal policy significantly. We have shown that an optimal policy within G^E should act according to the following two rules.

1) The cells are placed from the temporary buffer to the main buffer, high-priority cells first. If they do not fit, then low-priority cells are expelled starting from those closest to the head of the queue.
2a) If the cell at the head of the queue is of high priority, then it is served.
2b) If the cell at the head of the queue is of low priority, then either that cell is served, or all the low-priority cells from the head of the queue until the high-priority cell closest to the head of the queue are expelled and that high-priority cell is served.

Note that the two rules above characterize the optimal actions completely for some states and in general up to a binary decision of whether all low-priority cells at the head of the queue are dropped or none of them. As in the case of pushout policies, the above result holds for arbitrary arrival processes. Let G^{EO} be the class of policies which satisfy the above two rules. We claim that the optimal policy within G^E should belong to G^{EO}. More specifically we show the following.

Theorem 3.2: For every policy $\pi \in G^E$ there exists a policy $\tilde{\pi} \in G^{EO}$ such that if the system starts from the same initial state under the two policies, and the arrival process is identical under the two policies, we have

$$\tilde{D}^h(t) \leq D^h(t),$$

$$\tilde{D}^h(t) + \tilde{D}^l(t) \leq D^h(t) + D^l(t) \quad t = 1, 2, \cdots, \tag{11}$$

where $\tilde{D}^h, \tilde{D}^l(t)$ is the number of lost cells of high and low priority, respectively, under $\tilde{\pi}$ and similarly for $D^h(t), D^l(t)$ under π.

Note that in the theorem, $\tilde{D}^h(t) + \tilde{D}^l(t) \leq D^h(t) + D^l(t)$ implies $\tilde{D}^l(t) - D^l(t) \leq D^h(t) - \tilde{D}^h(t)$, which means even if $\tilde{D}^l(t) > D^l(t)$, the difference between them will be less than or equal to that between $D^h(t)$ and $\tilde{D}^h(t)$. Therefore, Theorem 3.2 implies the discarding cost in $\tilde{\pi}$ will be less than or equal to that in π. The proof of the theorem relies on the following lemma and is similar to that of Theorem 3.1. For a proof of the lemma the reader is referred to [13].

Lemma 3.2: For every policy $\pi \in G^E$ there exists a policy π_1, which acts according to rules 1 and 2 at slot 1 and is appropriately defined for $t = 2, 3, \cdots$, based on π, such that if the system starts from the same initial state and the arrival process is identical under the two policies, then we have

$$D^{1h}(t) \leq D^h(t),$$

$$D^{1h}(t) + D^{1l}(t) \leq D^h(t) + D^l(t) \quad t = 1, 2, \cdot$$

where $D^{1h}(t), D^{1l}(t)$ is the number of lost cells of high and low priority, respectively, under π_1.

IV. Interpretation Of The Results

The performance of a buffer allocation policy as far as the losses is concerned is completely characterized by the vector $(p_1, .., p_k)$ of the blocking probabilities p_i of each class $i = 1, .., k$. If the blocking probability vectors $(p_1, .., p_k)$ and $(\tilde{p_1}, .., \tilde{p_k})$ under two policies π and $\tilde{\pi}$ are related as $p_i \leq \tilde{p_i}, i = 1, .., k$ then policy π is better than policy $\tilde{\pi}$ under most performance criteria regarding losses. If the blocking probability vectors of two policies are not related in the sense that one is not strictly larger than the other, then the two policies are not directly comparable, and one may be preferable than the other depending on which traffic class has higher priority.

In Fig. 7 the region $ABCD'EFO$ represents the blocking probabilities achievable by any policy in the class of expelling policies, while the region $AA'O'F'FO$ represents the region of achievable blocking probabilities from policies of the discarding class. Any blocking probability pair on the lower boundary $AA'O'F'F$ of the discarding policies region has the property that is not strictly larger than any other blocking probability pair achievable by a discarding policy. When we are constrained in the discarding class of policies clearly we would like to operate the system at some point of the curve $AA'O'F'F$. Any discarding policy that is optimal with respect to a linear cost like the one considered in Section II will lie on the line $AA'O'F'F$, otherwise there will exist another policy with better performance with respect to that linear cost. The exact position of the point on the line depends on the selection of the coefficients c_1, c_2 in the linear cost. One interesting open problem is to find how can we achieve a point on $AA'O'F'F$ by appropriate selection of the coefficients.

As in the case of discarding policies, any blocking probability pair on $ABCD'EF$ has the property that it is not strictly worse than any other pair achievable by a policy in the expelling class. Therefore it is desirable to operate the system at some point of $ABCD'EF$.

The pushout class of policies is the subclass of the expelling policies with the property that the sum of the blocking probabilities of all classes is constant and equal to the minimum possible. The blocking probability pair of any policy in the pushout class will lie on the straight line segment CD'. Theorem 3.1 shows that the end points C and D' are achievable by the squeeze-out policy where classes 1 and 2 are given strict priorities, respectively. The points C' and D correspond to the pushout policies with priorities given to classes 1 and 2, respectively. The point O' corresponds to the policy at which no control is applied (a cell is discarded only if the buffer if full). This policy is also achieved by the optimal discarding policy where the blocking costs of the two classes are the same. Pushout policies do not achieve all the performance points achievable by expelling policies since it is possible to decrease the blocking probability of a class by blocking cells of the other class even before the buffer is full, something not allowed by pushout policies. The points in the segments $AA'BC$, $D'EF'F$ are achievable by the rest of the expelling policies. Theorem 3.2 shows that for any expelling policy with blocking probability pair (p_1, p_2), there is a policy in G^{EO} with blocking probability pair (p_1', p_2') such that $p_1 - p_1' \geq p_2' - p_2$. In other words, for the optimum expelling policy, the reduction of the blocking probability of the high-priority class is larger than the increase of the blocking probability of the low-priority class.

V. Numerical Results

Figs. 4–6 display some of the preliminary numerical results we have obtained. The objective was to compare the performance of some of the policies discussed in this paper. A two-priority system with i.i.d. arrivals was considered. The arrival process is derived from a binomial distribution and is the same as the one used in [11]. The arrival rate as well as the fraction of traffic from the two priority classes was varied. The cost of losing a high-priority cell was varied from 10 to 10^8 times the cost of losing a low-priority cell. Value iteration [12] was used to compute the performance of the optimal discarding, expelling and pushout policies as well as the default policy. The default policy is the one where cells are simply admitted to the main buffer in FIFO order, high-priority cells first, and dropped if it is full. We considered a system with main and temporary buffer sizes of 7 and 3, respectively. The squeeze-out and default policies corresponded to single points in the plots in Figs 4–6 since they are unaffected by the discarding costs. The performance of other pushout policies are provided for comparison. In both of these policies, low-priority cells from the temporary buffer do not push out low-priority cells from the main buffer but are dropped instead. In last-in-first-drop and first-in-first-drop (LIFD and FIFD) pushout policies high-priority cells pushout the low-priority cells that are, respectively, furthest from and closest to the head of the queue. Note that for most cases considered, there is little difference in the performance of the LIFD pushout, FIFD pushout and squeeze-out policies. As expected, the expelling policy performed better than the discarding policy; the difference between the two policies

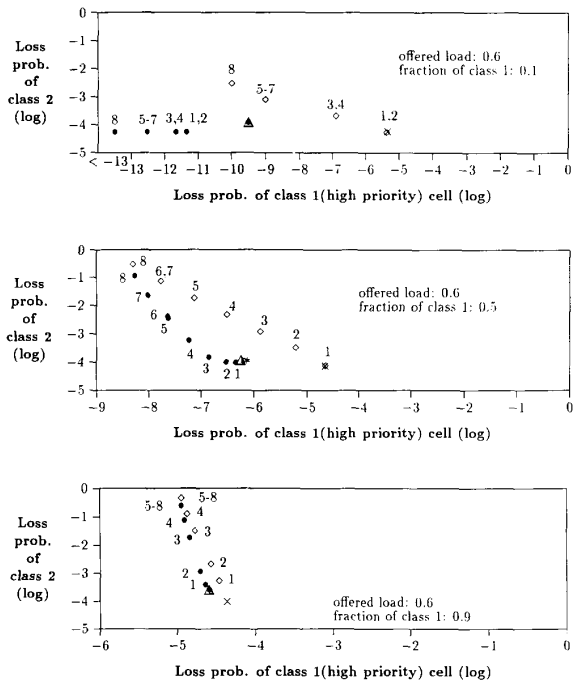

Fig. 4. Loss probabilities for two classes with main buffer size=7, temporary buffer size=3, discarding cost of a low-priority cell=1, discarding cost of high-priority cell changes from 10 to 10^8. In the figure "Δ" stands for the squeeze-out policy, "$*$" for LIFD pushout, "o" for FIFD pushout, "$\diamond$" for discarding policy, "$\times$" for default policy, which sets the thresholds of both classes to be the main buffer size, and "$\bullet$" for expelling policy. The numbers next to the "$\diamond$" and the "$\bullet$" stand for the powers of 10, of the discarding costs of class 1 cells used to obtain those points.

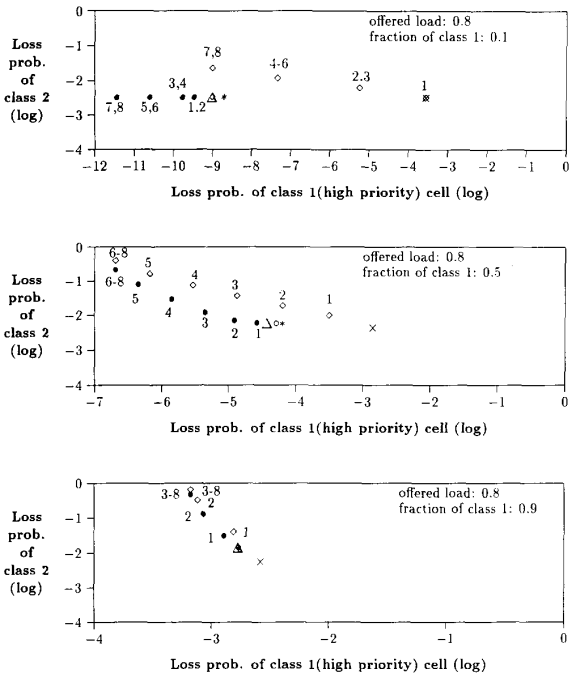

Fig. 5. Loss probabilities for two classes with main buffer size=7, temporary buffer size=3, discarding cost of a low-priority cell=1, discarding cost of high-priority cell changes from 10 to 10^8. In the figure "Δ" stands for the squeeze-out policy, "$*$" for LIFD pushout, "o" for FIFD pushout, "$\diamond$" for discarding policy, "$\times$" for default policy, which sets the thresholds of both classes to be the main buffer size, and "$\bullet$" for expelling policy. The numbers next to the "$\diamond$" and the "$\bullet$" stand for the powers of 10, of the discarding costs of class 1 cells used to obtain those points.

depended on the total traffic and relative proportions of the two classes of traffic. In many cases, changing the cost of dropping a high-priority cell by an order of magnitude or more, did not change the loss probabilities for high- and low-priority cells. We believe this was due to (a) the discrete nature of the problem and the small buffer size made the optimization "nonsmooth" and (b) increasing the loss of high-priority cells beyond a certain point made no difference to the optimum policy since it was already heavily biased in favor of high-priority cells. For the expelling policy, after examining the results of the value iteration algorithm it was determined that the decision of whether to serve the low-priority cell at the head of the queue or to serve the high-priority cell closest to the head of the queue was almost completely dependent on the number of high-priority cells in the main buffer. An expelling policy which made this decision based on a threshold on the number of high-priority cells in the main buffer achieved losses within 1.2% of the cost achieved by the optimal expelling policy. This sub-optimal policy could therefore be used as a basis for a practical implementation of the expelling policy.

For larger buffer sizes, a simulation was performed to compare the performance of the different policies. The results of the simulation are reported extensively in [4] and they are briefly summarized in the following. To closely simulate bursty traffic, 12 identical binary sources were considered generating arrivals. Each binary source consists of an "active" and an "inactive" state, and can be characterized by a two-state discrete time Markov chain. The traffic source alternates between these two states, and the number of time slots spent in each state is geometrically distributed. When in an "active" state the source generates cells in consecutive time slots, and in the "inactive" state no cells are generated. We call the duration of a busy period, i.e., the number of time slots that a source stays in the active state, a burst.

Each cell is generated by the above source is fed into a leaky bucket server to mark the cell priority. Tokens arrive at the leaky bucket deterministically. When the token pool inside the leaky bucket is full, newly arriving tokens are thrown away. An arriving cell that finds the token pool empty is marked as low priority. If there is at least one token in the permit pool, then the cell that obtains a token is marked as high priority, and the token count is decremented.

Unless stated otherwise, the main buffer size, is 100. We set the mean burst length $\ell = 20$ cells and the offered load $\rho = 0.9$ in most cases. The token pool size is set to the mean burst length. The token arrival rate is varied to change the mix of high- and low-priority cells. In all cases, ρ_1 represents the traffic load of high-priority cells. The proportion of low-priority cells is thus $(1 - \rho_1)/\rho$, where ρ is the aggregate offered load.

Four buffer management policies for two classes of cells were investigated. Namely, the discarding policy, FIFD (first-

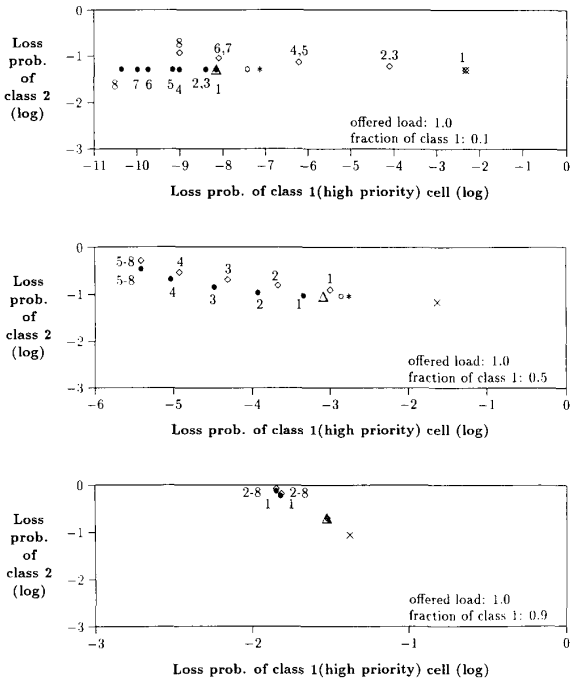

Fig. 6. Loss probabilities for two classes with main buffer size=7, temporary buffer size=3, discarding cost of a low-priority cell=1, discarding cost of high-priority cell changes from 10 to 10^8. In the figure "Δ" stands for the squeeze-out policy, "$*$" for LIFD pushout, "o" for FIFD pushout, "$\diamond$" for discarding policy, "$\times$" for default policy, which sets the thresholds of both classes to be the main buffer size, and "$\bullet$" for expelling policy. The numbers next to the "$\diamond$" and the "$\bullet$" stand for the powers of 10, of the discarding costs of class 1 cells used to obtain those points.

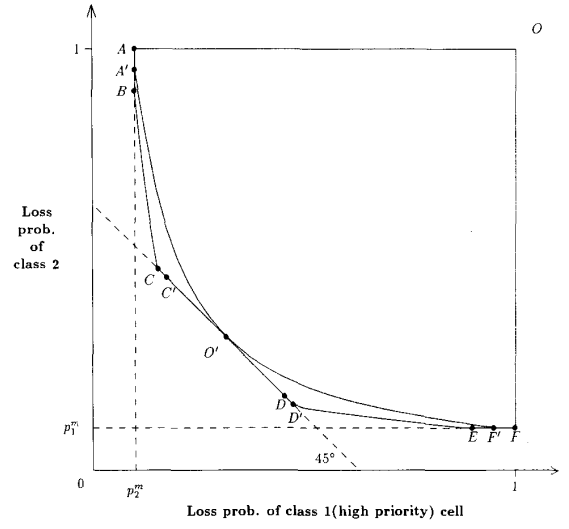

Fig. 7. The regions of achievable blocking probabilities for the different classes of policies are depicted

in-first-drop) pushout policy, LIFD (last-in-first-drop) pushout policy, and the expelling policy described earlier in this section. The optimal pushout policy, the squeeze out policy, is essentially the expelling policy with the threshold to expel

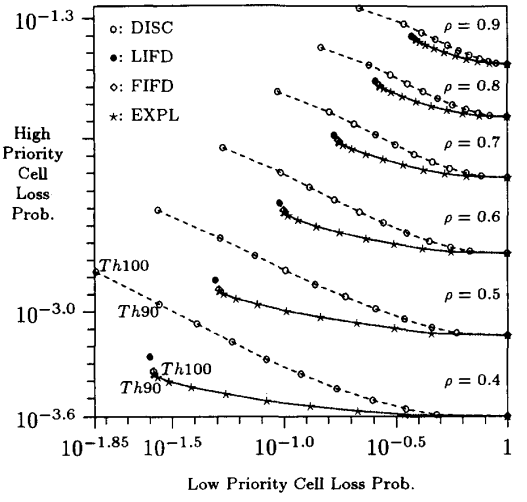

Fig. 8. High-priority cell losses versus low-priority cell losses. (The threshold varies from 100 to 0 cells, ρ from 0.4 to 0.9, $\rho_1/\rho = 0.9$, and mean burst length $\ell = 20$ cells).

equal to the buffer size. In both discarding and expelling policies, the lower the threshold value is, the better the high-priority cell performance we get. In other words, when the threshold value is small, more space is reserved for high-priority cells, thus the less the high-priority cell loss at the expense of low-priority cell loss. Both LIFD pushout and FIFD pushout perform similarly, with FIFD always slightly better than LIFD as we can see in Fig. 8. It is clear from the graph that the curves of the expelling policy cover the largest area from the upper right corner among all the policies that we discuss here. In other words, the region of performance for both high-priority cell losses and low-priority cell losses that can be achieved is broader with the expelling policy. Indeed, the expelling policy performs universally better than all the other policies discussed under the same traffic conditions. Fig. 8 also depicts that, as the offered load increases, the improvement of the expelling policy over the discarding policy decreases. This probably implies that the higher the total load is, the less we can do to improve the loss performance. However, we consider the expelling policy an attractive solution for real traffic, because we see further relative improvement when the losses go down to below 10^{-6}, which is the range for typical operations.

To provide acceptable QOS, the space control policy must give preference to the more loss-sensitive cells. It is thus clear that as the traffic mix ρ_1/ρ becomes lower, a better QOS can be assured. However, because the expelling policy favors the high-priority cells more than the other policies by selectively serving low-priority cells, the advantage of the expelling policy over the discarding policy is more significant when the traffic mix ρ_1/ρ is smaller (Fig. 9). Fig. 10 shows that the improvement of the expelling policy over the discarding policy is not very sensitive to the value of mean burst length as it varied from 10 to 30 cells. Figs. 11 and 12 show the performance gain as the buffer size is increased from 80 to 160 for two different traffic mixes. Again, the relative improvement

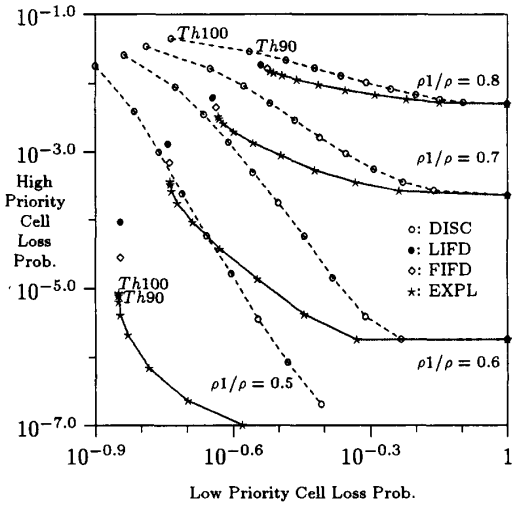

Fig. 9. High-priority cell losses versus low-priority cell losses. (The threshold varies from 100 (the buffer size) to 0 cells, ρ_1/ρ from 0.5 to 0.8, $\rho = 0.9$, and mean burst length $\ell = 20$ cells).

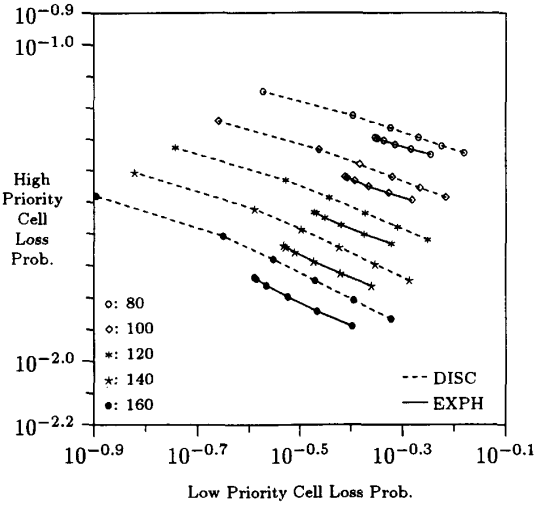

Fig. 11. High-priority cell losses versus low-priority cell losses. (The buffer size varies from 80 to 160 cells, $\rho_1/\rho = 0.9$, $\rho = 0.9$, and mean burst length $\ell = 20$ cells).

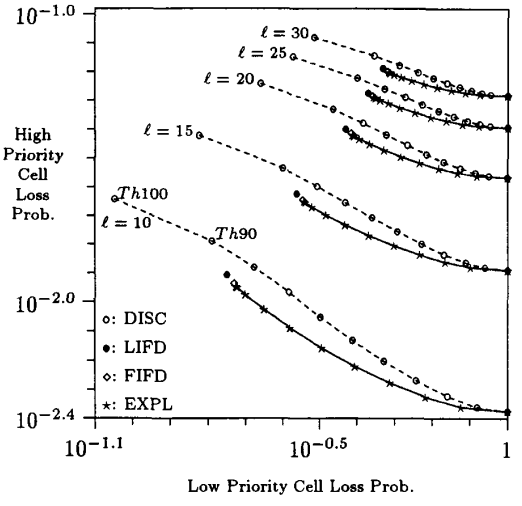

Fig. 10. High-priority cell losses versus low-priority cell losses. (The threshold varies from 100, the main buffer size, to 0 cells, ℓ from 30 to 10 cells, $\rho = 0.9$, and $\rho_1/\rho = 0.9$).

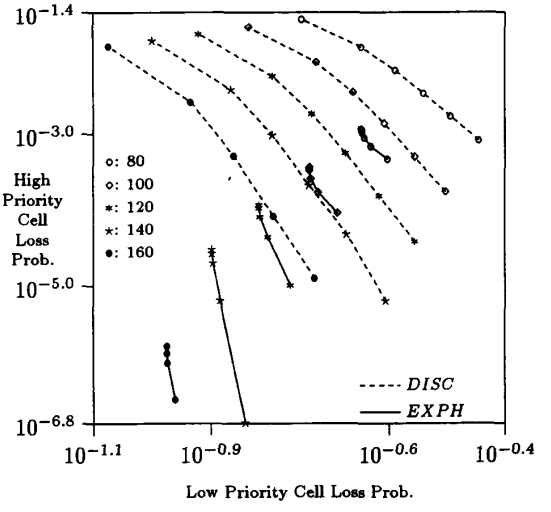

Fig. 12. High-priority cell losses versus low-priority cell losses. (The buffer size varies from 80 to 160 cells, $\rho_1/\rho = 0.6$, $\rho = 0.9$, and mean burst length $\ell = 20$ cells).

of the expelling policy increases for the case of the traffic mix corresponding to a larger proportion of low-priority cells.

VI. Discussion And Open Problems

The problem of buffer management at an output link of an ATM node was considered in the paper. Three classes of policies were studied and optimal policies with respect to losses were identified. The classes of policies that have been considered are implementable by the architectures proposed in [3] using the Sequencer chip.

Regarding the assumptions about the arrivals, for the expelling and pushout policies our results hold for any arrival process while for the discarding policies *i.i.d.* arrivals were assumed. If Markov-modulated arrivals are considered for the discarding class, then the state of the underlying Markov chain of the arrival process should be included in the state discription of the system in addition to the states of the main and temporary buffer. The discarding actions of the optimal policy in this case should rely on that underlying state as well.

Another open problem is the consideration of space priority policies for models corresponding to shared memory switches. In this case, cells destined to different switch outputs share a common buffer. Determining the optimal policies in this situation while maintaining cell loss fairness among cells destined to different output ports is a challenging problem. The ultimate goal remains to be the study of the buffer

management control schèmes as they interact at the network level in different nodes. This interaction determines the end-to-end system performance.

APPENDIX

Proof of Lemma 2.1: First we show that for every policy $g_0 \in G$ there exists a policy $g_1 \in G$ such that

$$V_{g_0}^{\beta}(\mathbf{x}_0) = V_{g_1}^{\beta}(\mathbf{x}_1).$$

Let the two systems have identical arrival processes and define g_1 to take the same actions with g_0 at every slot. Clearly, the same cells are dropped at every slot under both policies and the β-discounted cost is the same for $\mathbf{x}_0$ and $\mathbf{x}_1$. Note that since the state of the system is not the same under g_0 and g_1 the two policies are different in general. Nevertheless since the number of cells is the same at every slot under g_0 and g_1 it is indeed possible for g_1 to take the same actions as g_0. From the above argument clearly

$$V^{\beta}(\mathbf{x}_1) = \inf_{g \in G} V_g^{\beta}(\mathbf{x}_1) \le \inf_{g \in G} V_g^{\beta}(\mathbf{x}_0) = V^{\beta}(\mathbf{x}_0). \quad (12)$$

Similarly we can show

$$V^{\beta}(\mathbf{x}_1) \ge V^{\beta}(\mathbf{x}_0). \quad (13)$$

From (12), (13), the lemma follows. ◇

Proof of Theorem 2.2: Notice that for any action vector $\mathbf{a}_n$ and state $\mathbf{x} = (i, \mathbf{x}^T)$, we have

$$c(\mathbf{x}, \mathbf{a}_n) + \beta \textstyle\sum_{\mathbf{x}' \in \mathcal{X}} P_{\mathbf{x}\mathbf{x}'}(\mathbf{a}_n) V^{\beta}(\mathbf{x}')$$

$$= \sum_{j=n+1}^{B_T} c_{\mathbf{x}_j^T} + \beta \sum_{y \in \mathcal{X}^T} p_y V^{\beta}(1\{i > 0\}(i + n - 1) + 1\{i = 0\}(i + n), y). \quad (14)$$

In view of (14), in order to show (6), it is enough to show that

$$\min_{\mathbf{a} \in S(\mathbf{x})} \{c(\mathbf{x}, \mathbf{a}) + \beta \sum_{\mathbf{x}' \in \mathcal{X}} P_{\mathbf{x}\mathbf{x}'}(\mathbf{a}) V^{\beta}(\mathbf{x}')\} = \min_{n = 0, \cdots, B_T} \{c(\mathbf{x}, \mathbf{a}_n) + \beta \sum_{\mathbf{x}' \in \mathcal{X}} P_{\mathbf{x}\mathbf{x}'}(\mathbf{a}_n) V^{\beta}(\mathbf{x}')\}. \quad (15)$$

For any action vector $\mathbf{a}$ with n nonzero elements we have

$$c(\mathbf{x}, \mathbf{a}) + \beta \sum_{\mathbf{x}' \in \mathcal{X}} P_{\mathbf{x}\mathbf{x}'}(\mathbf{a}) V^{\beta}(\mathbf{x}') \ge c(\mathbf{x}, \mathbf{a}_n) + \beta \sum_{\mathbf{x}' \in \mathcal{X}} P_{\mathbf{x}\mathbf{x}'}(\mathbf{a}_n) V^{\beta}(\mathbf{x}'). \quad (16)$$

This is so because first we have $D(\mathbf{x}, \mathbf{a}) = D(\mathbf{x}, \mathbf{a}_n) = 1\{i > 0\}(i + n - 1) + 1\{i = 0\}(i + n)$, therefore

$$\beta \sum_{\mathbf{x}' \in \mathcal{X}} P_{\mathbf{x}\mathbf{x}'}(\mathbf{a}) V^{\beta}(\mathbf{x}') = \beta \sum_{\mathbf{x}' \in \mathcal{X}} P_{\mathbf{x}\mathbf{x}'}(\mathbf{a}_n) V^{\beta}(\mathbf{x}'). \quad (17)$$

Also, due to the decreasing priority arrangement of the cells in the temporary buffer, we can easily see that

$$c(\mathbf{x}, \mathbf{a}) \ge c(\mathbf{x}, \mathbf{a}_n). \quad (18)$$

From (17) and (18) we get (16), and that implies

$$\min_{\mathbf{a} \in S(\mathbf{x})} \{c(\mathbf{x}, \mathbf{a}) + \beta \sum_{\mathbf{x}' \in \mathcal{X}} P_{\mathbf{x}\mathbf{x}'}(\mathbf{a}) V^{\beta}(\mathbf{x}')\} \ge \min_{n = 0, \cdots, B_T} \{c(\mathbf{x}, \mathbf{a}_n) + \beta \sum_{\mathbf{x}' \in \mathcal{X}} P_{\mathbf{x}\mathbf{x}'}(\mathbf{a}_n) V^{\beta}(\mathbf{x}')\}$$

so (15) follows. ◇

Proof of Theorem 2.3: Let $u = g^{\beta}(\mathbf{x})$ in the following. Assume $u > 0$ initially. Since u minimizes the right-hand side of the dynamic programming equation, the difference of the right-hand side of (6) evaluated at $n = u - 1$ and at $n = u$ is

$$c_{\mathbf{x}_u^T} + \beta \sum_{y \in \mathcal{X}^T} p_y (V^{\beta}(i + u - 2, \mathbf{y}) - V^{\beta}(i + u - 1, \mathbf{y})) \ge 0.$$

We show first that $i + u \le t_{\mathbf{x}_u^T}$. If $i + u > t_{\mathbf{x}_u^T}$ then from Lemma 2.3 and the definition of the thresholds we get

$$0 \le c_{\mathbf{x}_u^T} + \beta \sum_{y \in \mathcal{X}^T} p_y (V^{\beta}(i + u - 2, \mathbf{y}) - V^{\beta}(i + u - 1, \mathbf{y}))$$

$$\le c_{\mathbf{x}_u^T} + \beta \sum_{y \in \mathcal{X}^T} p_y (V^{\beta}(t_{\mathbf{x}_u^T} - 2, \mathbf{y}) - V^{\beta}(t_{\mathbf{x}_u^T} - 1, \mathbf{y})).$$

If $t_{\mathbf{x}_u^T}$ is the threshold we should have

$$c_{\mathbf{x}_u^T} + \beta \sum_{y \in \mathcal{X}^T} p_y (V^{\beta}(i + u - 2) - V^{\beta}(i + u - 1, y))$$

$$= c_{\mathbf{x}_u^T} + \beta \sum_{y \in \mathcal{X}^T} p_y (V^{\beta}(t_{\mathbf{x}_u^T} - 2, y) - V^{\beta}(t_{\mathbf{x}_u^T} - 1, y)).$$

From the definition of $g^{\beta}(i, \mathbf{x}^T)$, if several terms achieve the minimum on the right-hand side of (9), the maximal value of i is used for t_j. This implies $t_{\mathbf{x}_u^T} = i + u$ which is a contradiction. Assume now that there is a $u' > u$ such that

$$i + u' \le t_{\mathbf{x}_{u'}^T}.$$

Note that $c_{\mathbf{x}_{u'}^T} + \beta \sum_{y \in \mathcal{X}^T} p_y (V^{\beta}(i + u' - 2, y) - V^{\beta}(i + u' - 1, y)) \ge 0$ from Lemma 2.3 and from the definition of threshold. We will show that this is a contradiction as well. Consider the terms

$$c_{\mathbf{x}_l^T} + \beta \sum_{y \in \mathcal{X}^T} p_y (V^{\beta}(i + l - 2, y) - V^{\beta}(i + l - 1, y)) \ge 0, \quad u \le l \le u'. \quad (19)$$

The nonnegativeness of the above terms is implied from Lemma 2.3 and the fact that the packets in the temporary buffer are stored in decreasing priority order. The difference of the right-hand side of (6) evaluated at $n = u$ and at $n = u'$ is

$$\sum_{l=u+1}^{u'} c_{\mathbf{x}_l^T} + \beta \sum_{y \in \mathcal{X}^T} p_y (V^{\beta}(i + u - 1, y) - V^{\beta}(i + u' - 1, y))$$

$$= \sum_{l=u+1}^{u'} (c_{\mathbf{x}_l^T} + \beta \sum_{y \in \mathcal{X}^T} p_y (V^{\beta}(i + l - 2, y) - V^{\beta}(i + l - 1, y)))$$

All the terms of the sum are positive from (19), therefore u cannot be the action of the optimal policy. If $u = 0$, we are interested only in the case $l(\mathbf{x}^T) > 0$. The proof is similar. ◇

Proof of Lemma 3.1: We construct the policy π_1 for $t = 2, 3, \cdots$ such that (10) holds. More specifically we show that for all $t = 1, 2, \cdots$ we have

$$D^{1l}(t) \geq \tilde{D}^l(t), \quad \text{(a)}$$

$$D^{1h}(t) + D^{1l}(t) = \tilde{D}^h(t) + \tilde{D}^l(t), \quad \text{(b)}$$

$$\sum_{i=j}^{B(t)} 1\{\mathbf{x}_i(t) = l\} + D^{1l}(t) \geq \sum_{i=j}^{\tilde{B}(t)} 1\{\tilde{\mathbf{x}}_i(t) = l\} + \tilde{D}^l(t), \quad j = 1, \cdots, B(t), \quad \text{(c)}$$

$$\text{and } \mathbf{x}_i(t) = \tilde{\mathbf{x}}_i(t), B(t) < i \leq B, \quad \text{(d)}$$

where $B(t)$ is the maximum buffer position at time t that contains one of the cells that were in the buffer at time $t = 1$. The variable $B(t)$ is defined at time t, just after the accepting and discarding decisions have been made. If at time t the buffer contains no cell that was in the system at time $t = 1$ then $B(t) = 0$. Because of the FCFS property of the policies considered, if $B(t) > 0$, then all packets in the buffer positions $1, .., B(t)$ were in the system at time $t = 1$. Notice that $B(1)$ is the same for all policies in G^p since a cell cannot be discarded if the buffer is not full. $\tilde{B}(t)$ is defined similarly to $B(t)$ for the system operated under policy $\tilde{\pi}$. The relations (c), (d) are auxiliary relations needed to show that (a), (b) hold for all t.

If all the cells in the temporary buffer fit into the main buffer without discards, then the state of the main buffer at $t = 1$ is identical under policies $\tilde{\pi}$ and π_1. If we let π_1 act identically to $\tilde{\pi}$ at $t = 2, \cdots$ then the states of the system under the two policies match and (a)-(d) hold trivially.

If all the cells do not fit in the main buffer without discarding then we proceed as follows. We show first that (a)-(d) hold for $t = 1$. Then we show that if (a)-(d) hold for $t = \tau$ and π_1 is appropriately defined at $t = \tau + 1$, based on the action that $\tilde{\pi}$ takes in the same slot, then (a-d) hold at slot $\tau + 1$ as well. By induction we conclude that policy π_1 can be defined in the slots $2, 3, \cdots$ such that (a)-(d) hold.

For $t = 1$ we have by definition $B(t) = \tilde{B}(t) = B$, hence (d) holds trivially. Also we can easily see that (b) holds. For relations (a), (c) we distinguish three cases, for the state of the system at $t = 1$, which are illustrated in Fig. 2.

In case 2, where the number of high-priority cells in the main and temporary buffer exceed the capacity of the main buffer, relations (a,c) easily follow since $D^{1l}(t)$ is equal to the total number of low-priority cells in the system.

In case 3, where all the low-priority cells that were in the main buffer at $t = 0$ are discarded at $t = 1$, relation (a) follows if we think that $D^{1l}(t)$ has the maximum possible value under any policy in G^p given that the main buffer is full at $t = 1$. Relation (c) follows if we see that the main buffer has the smallest possible number of low-priority cells under any policy and they are all at the end of the buffer.

In case 1, where only a fraction of the low-priority cells need to be pushed out, relation (a) holds for the same reason as in case 3. In order to see why (c) holds, we just need to observe that

$$\sum_{i=j}^{B(1)} 1\{\mathbf{x}_i(1) = l\} + D^{1l}(1) = L(0) - \sum_{i=1}^{j-1} 1\{\mathbf{x}_i(1) = l\} \quad \text{(c.1)}$$

$$\sum_{i=j}^{\tilde{B}(1)} 1\{\tilde{\mathbf{x}}_i(1) = l\} + \tilde{D}^l(1) = \tilde{L}(0) - \sum_{i=1}^{j-1} 1\{\tilde{\mathbf{x}}_i(1) = l\}, \quad \text{(c.2)}$$

where $L(0), \tilde{L}(0)$ are the total number of low-priority cells in the system at $t = 0$. Clearly

$$\sum_{i=1}^{j-1} 1\{\mathbf{x}_i(1) = l\} \leq \sum_{i=1}^{j-1} 1\{\tilde{\mathbf{x}}_i(1) = l\}. \quad \text{(c.3)}$$

Since according to π^{po} we push out the largest number of low-priority cells, and we start from the head of the queue, relations (c.1), (c.2), and (c.3) imply (c).

Now assume that relations (a)-(d) hold at τ. Define the policy π_1 at $\tau + 1$ based on the action of the policy $\tilde{\pi}$ at the same slot as follows. Push out $(B(\tau) - 1)^+ - \tilde{B}(\tau + 1)$ cells starting from the low-priority cells closest to the server that reside in positions $1, \cdots, B(\tau)$ and continue with the high-priority cells in the same positions. For the positions $B(\tau + 1), \cdots, B$ emulate $\tilde{\pi}$ at $\tau + 1$,, i.e., accept the same cells and put them in the same positions of the main buffer.

We argue in the following that the above construction is possible. Note first that $B(\tau) = \tilde{B}(\tau)$, therefore $\tilde{B}(\tau + 1) \leq (B(\tau) - 1)^+$ and the first part of the construction is possible. Furthermore note that the state of the buffer in positions $B(\tau), \cdots, B$ is identical under the policies $\pi_1, \tilde{\pi}$ and also the state of the temporary buffer is identical under the two policies. Hence π_1 can indeed emulate $\tilde{\pi}$ for the positions $B(\tau + 1), \cdots, B$ and the second part of the construction is feasible as well.

We argue in the following that (a)-(d) hold for $t = \tau + 1$. Clearly $B(\tau + 1) = \tilde{B}(\tau + 1)$ and (b), (d) follow immediately for $\tau + 1$. Relation (a) for $t = \tau + 1$ follows from relation (a) for $t = \tau$, the fact that the maximum possible number of low-priority cells are expelled under π^{po} from positions $1, \cdots, B(\tau)$ of the main buffer, and the fact that the main buffer is identical under the two policies in positions $B(\tau), \cdots, B$. Relation (c) for $t = \tau + 1$ follows from relation (c) for $t = \tau$, and the fact that the maximum possible number of low-priority cells are expelled under π^{po} starting from the cells closest to the head of the queue. This completes the induction step and the proof.◇

Proof of Theorem 3.1: We define inductively a sequence of policies π_k, $k = 1, 2, \cdots$ with the property that policy π_k acts identically to policy π^{po} for the first k slots. Policy π_1 has been defined in Lemma 3.1. Policy π_{k+1} is defined based on π_k as follows, Repeat the construction of Lemma 3.1 with policies π_k and π_{k+1} in place of $\tilde{\pi}$ and π_1 respectively and the construction starting at time $t = k + 1$. We can easily see

from Lemma 3.1 that we have

$$D^{(k+1)l}(t) \geq \tilde{D}^{kl}(t),$$

$$D^{(k+1)l}(t) + D^{(k+1)h}(t) = \tilde{D}^{kl}(t) + \tilde{D}^{kh}(t) \quad t = 1, 2, \cdots \tag{20}$$

where $D^{kh}(t), D^{kl}(t)$ are the numbers of dropped high and low-priority cells, respectively, by slot t when policy π_k schedules admissions. Under policy π_k the system evolves identically as under policy π^{po} until slot k. Therefore we have

$$D^l(k) = D^{kl}(k) \geq D^{(k-1)l}(k) \geq \cdots \geq \tilde{D}^l(k),$$

$$D^h(k) + D^l(k) = D^{kh}(k) + D^{kl}(k)$$

$$= D^{(k-1)h}(k) + D^{(k-1)l}(k) = \cdots = \tilde{D}^h(k) + \tilde{D}^l(k),$$

from (20) and Lemma 3.1.

◇

ACKNOWLEDGMENT

We would like to thank R. F. Chen for providing some of the simulation results and the reviewers who helped to considerably improve the presentation of the paper with their thorough reviews.

REFERENCES

[1] G. A. Awater and F. C. Schoute, "Optimal queueing policies for fast packet switching of mixed traffic," *IEEE J. Select. Areas Commun.*, vol. SAC-9, pp. 458-467, Apr. 1991.

[2] A. W. Berger, A. E. Eckberg, T. C. Hou, and D. M. Lucantoni, "Performance characterizations of traffic monitoring, and associated control mechanisms for broadband "packet" networks," in *Proc. IEEE GLOBECOM 90*, San Diego, CA, Dec. 1990, pp. 350-354.

[3] H. J. Chao and N. Uzun, "A VLSI sequencer chip for ATM traffic shaper and queue manager," *IEEE J. Solid-State Circuits*, pp. 1634-1643, Nov. 1992.

[4] R. F. Chen and S. Panwar, "Optimal buffer management for ATM congestion control," Polytechnic Univ., Tech. Rep. CATT-94-74, 1994.

[5] A. Gravey and G. Hebuterne, "Mixing time and loss priorities in a single server queue," in *Proc. 13th ITC*, Copenhagen, June 1991, pp. 147-152.

[6] H. Kroner, "Comparative performance study of space priority mechanisms for ATM networks," in *IEEE INFOCOM 90*, San Francisco, CA, June 1990, pp. 1136-1143.

[7] H. Kroner, G. Hebuterne, P. Boyer, and A. Gravey, "Priority management in ATM switching nodes," *IEEE J. Select. Areas Commun.*, vol. 9, pp. 418-427, Apr. 1991.

[8] S. A. Lippman, "Applying a new device in the optimization of exponential queuing systems," *Oper. Res.*, vol. 23, no 4, pp. 687-710, July-Aug. 1975.

[9] D. W. Petr and V. S. Frost, "Optimal packet discarding: An ATM-oriented analysis model and initial results," *IEEE INFOCOM 90*, San Francisco, CA, June 1990, pp. 537-542.

[10] ——, "Priority cell discarding for overload control in BISDN/ATM networks: An analysis framework," *Int. J. Digital and Analog Commun. Syst.*, vol. 3, no. 2, Apr.-June 1990, pp. 219-227.

[11] ——, "Nested threshold cell discarding for ATM overload control: optimization under cell loss constraints," *IEEE INFOCOM 91*, FL, Apr. 1991, pp. 1403-1412.

[12] S. Ross, *Introduction to Dynamic Programming*. New York: Academic, 1983.

[13] L. Tassiulas, Y. C Hung, and S. Panwar, "Optimal buffer control during congestion in an ATM network node," Polytechnic Univ., Tech. Rep. CATT-93-66, 1993.

[14] Y. Yeh, M. G. Hluchyj, and A. S. Acampora, "The Knockout Switch: A simple, modular architecture for high-performance packet switching," *IEEE J. Select. Areas Commun.*, vol. SAC-5, pp. 1274-1283, Oct. 1987.

Leandros Tassiulas (S'89–M'91) was born in 1965, in Katerini, Greece. He received the Diploma in electrical engineering from the Aristotelian University of Thessaloniki, Thessaloniki, Greece, in 1987, and the M.S. and Ph.D. degrees in electrical engineering from the University of Maryland, College Park, in 1989 and 1991, respectively.

Since September 1991, he has been in the Department of Electrical Engineering at Polytechnic University, Brooklyn, NY, as an Assistant Professor. His research interests are in the field of computer and communication networks with emphasis on wireless communications and high-speed networks, in control and optimization of stochastic systems and in parallel and distributed processing.

Yao Chung Hung received the B.S. degree in computer science from Tunghai University, Taichung, Taiwan, in 1985. He also received the M.S. degree in systems engineering in 1989 and the Ph.D. degree in electrical engineering in 1993 from the Polytechnic University, Brooklyn, NY.

He was a graduate assistant at Polytechnic University working on real-time scheduling policies and optimal buffer management policies. He has been with Allerion, Inc., East Hanover, NJ, since 1993 working on network management.

Shivendra S. Panwar (S'82–M'85) was born in Delhi, India, on December 15, 1959. He received the B.Tech. degree in electrical engineering from the Indian Institute of Technology, Kanpur, in 1981, and the M.S. and Ph.D. degrees in electrical and computer engineering from the University of Massachusetts, Amherst, in 1983 and 1986, respectively.

From 1981 to 1985 he was a Research Assistant at the University of Massachusetts. He joined the Department of Electrical Engineering at the Polytechnic University, Brooklyn, NY, where he is now an Associate Professor. He is currently an Associate Director of the New York State Center for Advanced Technology in Telecommunications. He spent the summer of 1987 as a Visiting Scientist at the IBM Thomas J. Watson Research Center, Yorktown Heights, NY, and has been a Consultant to AT&T Bell Laboratories, Holmdel, NJ. His research interests include the performance analysis and design of high-speed networks.

Dr. Panwar is a member of Tau Beta Pi and Sigma Xi. He has served as the Secretary of the Technical Affairs Council of the IEEE Communications Society from 1992 to 1993 and is a member of the Technical Committee on Computer Communications. He was the Co-Chairman of the Technical Program Committee and a member of the Organizing Committee of the Second IEEE Network Management and Control Workshop, Tarrytown, NY, September 1993. He has also been a member of the Technical Program Committee of INFOCOM '90 and '93, and a Session Chair in INFOCOM '90. He is co-editor of the forthcoming book, *Network Management and Control, Vol. II* which will appear in 1994.