Optimal Buffer Sharing in a Shared Memory Packet Switch or Demultiplexer *

Rajarshi Roy and Shivendra S. Panwar †
Center for Advanced Technology in Telecommunications
Polytechnic University
6 Metrotech Center
Brooklyn, NY 11201

January 7, 1997

Abstract

In this paper we study the problem of the optimal design of buffer management policies for a shared memory multiport device such as an ATM switch or demultiplexer. A system with cells with two different space priorities is considered. Our objective is to find the optimal policy that minimizes the total weighted cell loss. The problem of finding the optimal policy within the class of *pushout* policies is considered. Using sample path techniques the search space for the optimal policy is reduced to a subset of *pushout* policies. A numerical study based on the value iteration technique is used to further investigate the structure of the optimal policy.

Keywords: Markov Decision Theory, Sample path techniques, Shared Memory Switch, Buffer Management, ATM.

1 Introduction

In a shared memory fast packet switch the entire switch fabric memory may be shared by all the output port queues. A similar situation may arise in a shared memory demultiplexer. In this paper we present the mathematical formulation of the optimal buffer management problem for such devices and present some analytical and numerical results. Our goal is to design optimal policies within the *pushout* class of policies which yield minimum weighted cell loss.

*This research was supported by the New York State Center for Advanced Technology in Telecommunications, Polytechnic University.

†Tel: 718 260 3740, Fax: 718 260 3074, e-mail: panwar@kanchi.poly.edu

There has been considerable amount of prior work in this area. The queueing analysis of different buffer sharing schemes and their relative merits was analyzed in [1]. The problem of designing optimal policies for the purpose of optimizing certain performance criteria is considered in [2]. They only consider the problem of searching for optimal policies within the class where cells can only be blocked at the entry point of the buffer; dropping cells once it is accepted in the switch is not allowed. In [3] *pushout* is allowed and new arriving cells can push out cells from the longest logical queue. This policy turns out to be optimal only for a symmetric system. Guerin, Cidon, Georgiadis and Khamisy [4] considered a Poisson arrival, single loss priority and exponential service model for sharing memory in a switch with two output ports. They have established the optimality of a threshold based pushout scheme using Markov decision theory.They also calculated the values of those thresholds. For a N ported system, $N \geq 3$, they have results for balanced input and equal service rates at all the output ports. The non-optimality of the work of Hsaio [3] for the case of unequal service rates in a two ported system is shown in their work. For an N-ported system they have results only for the symmetric case.

Hung et al in [5] have provided optimal policies within the class of *discarding*, *pushout* and *expelling* policies using dynamic programming and sample path based techniques. They consider the case of general arrival models for the case of single output ports.

Hahne and Choudhury [6] have proposed a set of pushout and buffer sharing rules. They proposed a back-pressure mechanism for sharing buffers across switch stages and a pushout scheme for sharing buffers among different logical queues sharing a common memory. They have also applied their policy to traffic with multiple priorities. The lowest priority cell at the head of the queue is selected for pushout. They have applied this work for hierarchical switch system and Banyan based switch system [6], [7]. Their simulation study reveals that the "Delayed Pushout" scheme works well under all load conditions.

Here we model the the switch fabric itself as a buffer of finite capacity which is being shared by several logical queues, each of which may contain high and low priority cells. Our model is described in detail below.

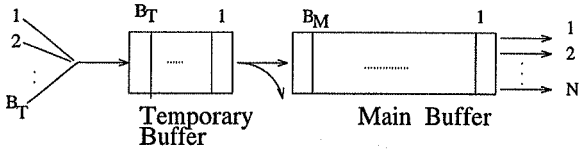

Figure 1: The system model

2 Model and buffer policy descriptions

In our model, the two parts in which total memory of the switch/demultiplexer B is subdivided are the main buffer with capacity B_M, and the temporary buffer with capacity B_T. It is a N ported, slotted system with cells destined to each output port or server constituting a logical queue. FIFO order is to be maintained within each of the logical queues once the cells are accepted in the main buffer, each of which contains high and low priority cells. Service time is deterministic and the servers serve one cell in every time slot if such a cell is available. In one slot at most B_T cells can arrive. Clearly for a switch $N = B_T$, if at each input line at most one cell can arrive in every time slot. The value of B_T in case of a demultiplexer is dependent on the input line speed. At every decision epoch cells from the temporary buffer are either admitted in the main buffer, possibly by pushing out some cells from the main buffer, or dropped.

The class of *pushout* policies G_P is defined by the following rules: (a) A cell can be expelled from main buffer only if it is pushed out by another cell in the temporary buffer and the main buffer is full, (b) A cell from the temporary buffer can be discarded only if the main buffer is full. (c) A cell can never be dropped if there is room for it in the main buffer.

The class of *squeeze-out* policies G_{P_1} is defined as the following. Without loss of generality, let us assume that at every slot a given number of buffer spaces is allocated to each logical queue. Then under G_{P_1} policies the rules are to be followed are: (a) Append the cells that are in the temporary buffer and belong to logical queue n to the end of the main buffer in the allocated position, high priority cells first (if switch model) or in FIFO order (if demultiplexer model), (b) If the amount of buffer allocated for logical queue n in the main buffer is full, and there are cells of that logical queue in the temporary buffer, push out the low-priority cells starting from those closest to the head of that logical queue, (c) If all low priority cells of that logical queue which are in the main buffer are pushed out, discard all the remaining cells of that logical queue which are in the temporary buffer.

The amount of buffering allocated to logical queue n at time t is $B_n(t)$. It is evident that $B_n(t) \leq B_M$.

3 The MDP problem formulation

The cells are classified into two priority classes. The high priority classes are more sensitive to cell losses. Without loss of generality we assume that the priority of class 1 is higher than priority of class 2. The priority of a class is reflected by the cost that is incurred by the dropping of a cell of that class. We denote by $X_{n,i}(t)$ the class of the cell residing at the i-th place of the n-th logical queue by the end of the slot $[t-1,t]$. Here $i = 1, 2, ..., L_n(t)$, where $L_n(t)$ is the number of buffer places the logical queue n occupies at t^-. Clearly, $\sum_{n=1}^{N} L_n(t) = B$. $X_1(t), X_2(t), ..., X_N(t)$ is the vector that can be taken as a state, where $X_n(t) = (X_{n,i}(t) : i = 1, 2, ..., L_n(t))$. If there is any empty buffer space, it can be arbitrarily assumed to be part of a particular fixed logical queue and the value of $X_{n,i}(t)$ corresponding to that is 0. Thus $X_{n,i}(t)$ can take on the values 0, 1 or 2.

We want to find out the policy π_1 amongst the class of *pushout* policies G_P so that $E_{\pi_1}(\sum_{t=0}^{T}(C_h \cdot D_h(t) + C_l \cdot D_l(t))|\text{any initial state } = x) \leq E_{\pi_2}(\sum_{t=0}^{T}(C_h \cdot D_h(t) + C_l \cdot D_l(t))|\text{any initial state } = x)$, $\forall \pi_2 \in G_P$, T is the horizon size. Here, $D_h(t)$ and $D_l(t)$ are the number of high and low priority cells dropped from the system, respectively up to the end of slot $[t-1,t]$. C_h and C_l are positive real numbers, $C_h \geq C_l$. E_{π_i} is the expectation when policy π_i is used.

4 Some Results

The objective of theorem 1 which is stated below is to show that the policy π_{p_0} which is optimal in G_P in the sense that $D_h^{\pi_{p_0}}(t) + D_l^{\pi_{p_0}}(t) \leq D_h^{\pi}(t) + D_l^{\pi}(t)$, $\forall$ integer time $t \in \{0, 1, 2, 3, ...\}$ and $\forall \pi \in G_P$ such that $\pi \neq \pi_{p_0}$, and also $D_h^{\pi_{p_0}}(t) \leq D_h^{\pi}(t)$, lies in $G_{P_1} \subset G_P$. Here, $D_h^{\pi}(t)$ and $D_l^{\pi}(t)$ are the number of high priority and low priority cells dropped up to time t, under policy π, respectively.

Note that in a switch model the cells that came in one time slot are coming from different sources, because an input line can admit at most one cell in every time slot. So here the reordering of cells as they are shifted from the temporary to the main buffer is allowed. But in a demultiplexer model there can be one high speed source that generates more than one cell destined to one output port in one time slot. In this case the reordering of cells while putting them from temporary to the main

buffer is not allowed. Once the cells are transferred to the main buffer we maintain FCFS policy within every logical queue, since they may belong to the same virtual circuit.

Theorem 1: For the same initial state, and the same set of arrivals, under policies π_1 and π_2, where $\pi_1 \in G_{P_1}$ and $\pi_2 \in G^c_{P_1} \cap G_P$, we have $D_h^{\pi_1}(t) \leq D_h^{\pi_2}(t)$ and $D_h^{\pi_1}(t) + D_l^{\pi_1}(t) = D_h^{\pi_2}(t) + D_l^{\pi_2}(t)$, $\forall t \in \{1, 2, ...\}$.

The proof of this theorem depends on Lemma 1.1 which is stated below.

Lemma 1.1: For any policy $\pi_2 \in G^c_{P_1} \cap G_P$, there exists a policy π_3, which acts as a policy that belongs to G_{P_1} at $t = 1$, and if appropriately defined for all integer time $t \geq 1$, under the same initial conditions and the same arrivals, $D_h^{\pi_3}(t) \leq D_h^{\pi_2}(t)$ and $D_h^{\pi_3}(t) + D_l^{\pi_3}(t) = D_h^{\pi_2}(t) + D_l^{\pi_2}(t)$, $\forall t \in \{1, 2, ...\}$.

The proofs of Lemma 1.1 and Theorem 1 are not included for the sake of brevity. The proof techniques are similar to those employed by us in [5].

Thus for any policy $\pi' \in G_P$, by Theorem 1, there exists a policy $\pi_1 \in G_{P_1}$ such that $D_l^{\pi_1}(t) \geq D_l^{\pi'}(t)$ and

$D_h^{\pi_1}(t) + D_l^{\pi_1}(t) = D_h^{\pi'}(t) + D_l^{\pi'}(t)$

$0 \leq D_l^{\pi_1}(t) - D_l^{\pi'}(t) = D_h^{\pi'}(t) - D_h^{\pi_1}(t)$

$0 \leq C_l(D_l^{\pi_1}(t) - D_l^{\pi'}(t)) \leq C_h(D_h^{\pi'}(t) - D_h^{\pi_1}(t))$

$0 \leq C_l \cdot D_l^{\pi_1}(t) + C_h \cdot D_h^{\pi_1}(t) \leq C_l \cdot D_l^{\pi'}(t) + C_h \cdot D_h^{\pi'}(t)$

So, $E_{\pi_1}[\sum_{t=0}^{N}(C_h \cdot D_h(t) + C_l \cdot D_l(t))|\text{any initial state} = x] \leq E_{\pi'}[\sum_{t=0}^{N}(C_h \cdot D_h(t) + C_l \cdot D_l(t))|\text{any initial state} = x]$.

5 Some Counterexamples

The following are two counterexamples to sample-path dominance conjectures. However, these conjectures hold true in the expected value sense at least for some values of the load parameters. That we checked using the numerical procedure outlined in section 6.

5.1 Counterexample 1

This is a counterexample to the conjecture that a high priority cell should always push out a low priority cell, if there is any. Here it will be shown that a performance criteria such as given the previous section cannot be satisfied for every sample path of the input process if such a policy is followed. We consider a 2x2 switch with four main buffer places and two temporary buffer places. We refer to the policy that recommends push out from the queue with a low priority cell as the conjectured policy and the other one as the alternative policy. We start with the same initial state with one low priority cell destined to output 1, three high priority cells destined to output 2 and two high priority cells in the temporary buffer destined to output 2. Under the conjectured policy the low priority cell is dropped and one of the high priority cells are accepted from the temporary buffer. Under the alternative policy both the high priority cells are dropped from the temporary buffer. Now if two high priority cells arrive in the next time slot, then at the end of the second slot we have the same state under both the policies. If in all the future decision epochs we follow the same actions for both the policies, then under the alternative policy we incur one less low priority cell loss.

5.2 Counterexample 2

This is a counterexample of the conjecture that a low priority cell should never push out a high priority cell under under a sample path-wise optimal policy. Again, consider a system with a main buffer size of four and temporary buffer size of two. The initial state is such that there are four high priority cells destined to output 2 in the main buffer and one low priority cell destined to output 1 in the temporary buffer. Under the conjectured policy we drop the low priority cell and under the alternative policy we push out one high priority cell from the main buffer and admit the low priority cell. Under the conjectured policy we serve one high priority cell destined to output 2 and under the alternative policy we serve the low priority cell destined to output 1 and one high priority cell destined to output 2. If in the next slot two high priority cells destined to output 2 arrive, then at the end of that slot we enter the same state under both the policies. In future decision epochs both the policies can take the same actions at every slot. So the total cell loss under the alternative policy will be less.

These counterexamples do show that the inter-queue push out policy investigation is more complex and the results are not sample path-wise true but may be true in the expected value sense. We therefore performed a numerical study using value iteration since the goal of this ongoing work is to find out the structure of the optimal policy using the numerical study as a supporting tool.

6 Numerical study

We considered a two-ported shared memory switch modelled as a queue with bounded buffer and two servers each with constant service rate of one time slot/cell. For the numerical study we have assumed that at any time slot 0, 1 or 2 cells can arrive with probabilities a_0, a_1 and a_2 respectively. The sum of these probabilities is 1. The ar-

riving cells can be routed to one of the two logical queues with probabilities b_1 and b_2, $b_1 + b_2 = 1$. A cell is of high or low priority with probability c_1 or c_2 (d_1 or d_2), conditioned on the event that the cell is destined to queue 1 (2). We have here $c_1 + c_2 = 1$ and $d_1 + d_2 = 1$. If we drop a high priority cell we incur a loss of 1000 and if we drop a low priority cell we incur a cost of 1 (i.e., $C_h = 1000$ and $C_l = 1$). Our system has six buffer places of which two are temporary buffer places. At every decision epoch we have to make sure that after the cell dropping/pushout decision is taken we should not be left with more than four cells in the buffer because at most two cells can come between this decision epoch and the next one. Theorem 1 of section 4 shows that once we have decided to drop cells from a particular logical queue we start with the low priority cells at the head of that logical queue. Then we form the problem in the framework of Markov decision theory and seek to find out the optimal policy numerically that minimize the total undiscounted expected cost over an infinite horizon. We use $max_{(i \in S)}|V^{N+1}(i) - V^N(i)| \leq 0.0001$, where S is the state space and $V^N(i)$ is the value of the value function for state i with a horizon size of N, as our stopping criteria for convergence.

The numerical results reveal that pushout from the longest queue policy when we have only either high or low priority cells in the buffer is not optimal for an unbalanced load. Rather, the logical queues have thresholds k_1 and k_2 with their sum at least equal to B. Thus if we need to drop cells we should check which logical queue has exceeded its threshold, and if the arriving cells belong to that one then they get dropped. If it is the other one then these new cells are accepted and cells are pushed out from the other logical queue. For example, for a balanced load when there are 3 cells of high priority for both the logical queues numerical computation shows that we should drop one from each logical queue. But for case of 70 percent of the incoming traffic going to output port 1, we observed that the optimal policy dropped 2 cells from logical queue 1. This clearly shows the change in the value of thresholds with the offered load.

Numerical computations did show that we have to drop a low priority cell of either logical queue whenever we need to admit a new arriving high priority cell. So our counterexamples which are sample-pathwise true for some sample paths do not hold in an expected sense, at least for the values of the parameters we consider.

If two states has the same number of high and low priority cells for both the logical queues but in the first state the low priority cells are ahead in their logical queues then the value of $V^N(i)$ for the first state will be higher than that of the second state. This result is in accordance with Theorem 1, which states that optimal policy within the class G_P lies in $G_{P_1} \subset G_P$.

Other structural properties of the optimal policy are being currently investigated.

References

[1] F. Kamoun and L. Kleinrock, "Analysis of Shared Finite Storage in a Computer Network Node Environment Under General Traffic Conditions," *IEEE Trans. on Communications*, Vol. COM-28, No. 7, July 1980.

[2] G. J. Foschini and B. Gopinath, "Sharing Memory Optimally," *IEEE Trans. on Communications*, Vol. COM-31, No. 3, March 1983.

[3] S. X. Wei, E. J. Coyle and M. T. Hsiao, "An Optimal Buffer Management Policy for High Performance Packet Switching," *IEEE GLOBECOM'91*, Vol. 2, pp. 924-928, December, 1991.

[4] I. Cidon, L.Georgiadis, R. Guerin, A. Khamisy, "Optimal buffer sharing," *IEEE JSAC*, September 1995, vol 13, No. 7, pp. 1229-1240.

[5] L. Tassiulas, Y. C. Hung and S. S. Panwar, "Optimal Buffer Control During Congestion in an ATM Network Node," *IEEE/ACM Transactions on Networking*, August 1994, Vol. 2, No. 4, pp. 374-386.

[6] Abhijit Choudhury, Ellen L. Hahne, "Buffer Management in a Hierarchical Shared Memory Switch," *IEEE INFOCOM'94*, Vol. 3, pp. 1410-1419, June, 1994.

[7] Debasis Basak, Abhijit K. Choudhury, Ellen L. Hahne, "Sharing Memory in Banyan-based ATM Switches," Preprint.