A Simple and Fast Scheduler for Input Queued ATM Switches

Hyojeong Song, Lillykutty Jacob
Dept. of Comp. Sci., KAIST, Taejon 305-701, Korea
{hjsong,lilly}@camars.kaist.ac.kr

Hyun-Gon Kim
LG Information & Communications Ltd., Anyang 431-080, Korea

Boseob Kwon
Electronics and Telecommunications Research Institute, Taejon 305-350, Korea

Jai-Hoon Chung
Samsung Electronics Co., Ltd., Seoul138-240, Korea

Hyunsoo Yoon
Dept. of Comp. Sci., KAIST, Taejon 305-701, Korea

Abstract

Many 'output-scheduling' algorithms have been proposed for improving the performance of input queued asynchronous transfer mode (ATM) switches, whereby cells from different random-access input queues destined for the same output can be scheduled for non-conflicting transmissions. An optimal output-scheduling algorithm, one with the full coordination of transmissions to all outputs, can approach the performance of output queueing. Because of the complexity of such an optimal scheduler, output schedulers proposed in the literature are without such coordination. We propose a simple way to incorporate such a full coordination in output-scheduling with much simple hardware. Throughput of the input queueing switch thus approaches that of the output queueing switch, without speed-up, input/output grouping or complicated hardware. To make the output-scheduling algorithm fast enough, we incorporate parallelism and pipelining. We perform detailed simulation study of the performance of the input queueing switch with the proposed scheduling algorithm.

1. Introduction

For an internally nonblocking ATM switch, buffering is required to resolve the contention that occurs when multiple ATM cells simultaneously arrive at different input ports destined for the same output port. Only one cell at a time (i.e., in each time slot) can be transmitted over an output link; the rest must be temporarily stored in a queue. Output queueing yields the best performance possible in a switch: each input/output line of an output queued switch can be loaded upto 100%. However, the hardware cost of output queueing is prohibitive for all but the smallest switches. We restrict our attention to input queued switches.

It is well known that when FIFO queues are used, the throughput of an input queued ATM switch with independent and uniform input traffic can be limited to just 58.6% when the number of inputs/outputs N is large [1]. When arrivals are correlated, the throughput can be even lower [2]. The throughput is limited because a cell can be held up by another cell queued ahead of it that is destined for a different output. This phenomenon is known as HOL blocking.

Numerous papers have indicated that by using non-FIFO (random-access) input queues and using good scheduling policies, effect of HOL blocking can be reduced and much higher throughputs can be obtained [1, 3, 4, 5, 6, 7]. With random-access queues, an input may transmit to any one of the outputs for which it has a queued cell, with the constraint that each input can send at most one cell and each output can receive at most one cell, in each time slot. Random access input queues lead to forwarding cells through the switch fabric in an order different from the order in which they arrived. However, the switch maintains the cell sequence within each 'flow', since only the first queued cell in each flow is eligible to be transmitted through the fabric; where a flow is a stream of cells between a given input-output pair.

The difficulty is in devicing an algorithm that is both fast enough to schedule cells at high link speeds and efficient enough to deliver high link throughput, combined with minimal hardware complexity. Hluchyj et al. [1] suggested an iterative method in which an input that loses the first round of the contention tries with the second cell in its queue on the second round, and so on. After some number of iterations k (called window size), the winning cells from the various inputs are sent through the switch fabric to the outputs. This scheme has a growth problem because it is a sequential procedure, meaning that the speed of the algorithm is proportional to both the number of input ports and the number of iterations. In the parallel iterative matching (PIM) algorithm [3] and in the SLIP-IRRM (iterative round robin matching) algorithm [4], all inputs and outputs perform iterations in parallel, where each iteration involves three phases: request, grant and accept phases. The request, grant, and accept proptocol is implemented by running a wire between every input and output, which requires scheduling logic hardware that grows as $O(N^2)$. Also, though these algorithms provide high throughput (asymptotically 100%) for independent and uniform traffic, they perform less well and are unable to sustain high throughput if the traffic is not uniformly distributed over outputs, or if the arrival process is not Bernoulli.

Regarding efficiency, an optimal scheduling algorithm could approach the performance of output queueing but would require coordination of transmissions to all the outputs. Because of the complexity of such an optimal scheduler, algorithms have been proposed without such coordination, i.e., each output is allowed to schedule the transmissions independently of the others, thus leading to much simplified hardware [5, 6, 7].

Our approach is to use advance reservation (i.e., reservation in an earlier time slot than the one in which the cell is actually transmitted to the output port), parallelism and pipelining inorder to achieve the performance of optimal scheduler (with full coordination of transmissions to all the outputs), that is both fast enough to schedule cells at high link speeds and less complicated to be implemented. Fig. 1 shows a simplified block diagram of an input queueing ATM switch [7, 8]. The switch is composed of three functionally independent components: the input buffer (IB) modules, the contention resolution (CR) module, and a self-routing nonblocking space-division switch.

An approach incorporating both advance reservation and pipelining was used in [8] for scheduling in an ATM switch of the same configuration as Fig. 1. The basic scheme proposed there is unfair in the sense that the throughputs from various input ports differ by significant amounts. To solve this fairness problem, a 'barrel-shifter' is used to cycle the sequence in which the inputs are scheduled, which makes the implementation more complex. Also, switch throughput, cell delay and cell loss rate are all sensitive to the parameter M, the frequency with which the barrel-shifter is operated (called rotation period).

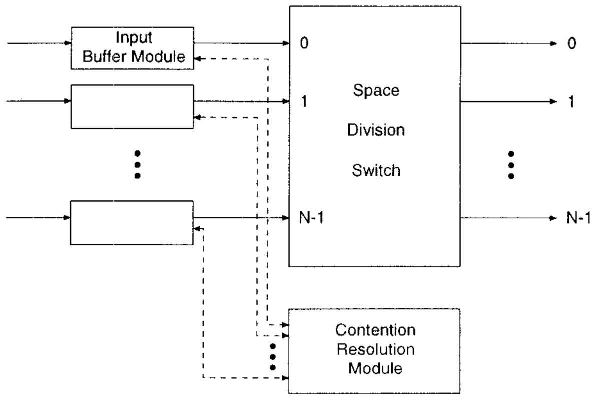

Figure 1. Simplified block diagram of an input queueing ATM switch

The scheduler proposed in this paper has inherent fairness property without any additional hardware such as barrel-shifter. Also, it is fast and highly efficient in the sense that it provides as high a throughput as 96% for feasible implementations. In Section 2 the different aspects of the scheduler are described in detail. Section 3 presents the details of the simulation study with the new scheduler, and Section 4 gives the conclusion.

2. Cyclic Reservation Interval Scheduler

In the detailed architecture of the switch with a configuration as shown in Fig. 1, each IB module consists of a random-access buffer, a send buffer and an IB controller [8]. ATM cells that arrive at each IB, in a synchronous, time slotted fashion, are stored in the order of their arrival, in the random access buffer. Using information from the IB controllers, the CR module schedules, in advance, the transmission times of cells in the random access buffers. Each scheduled cell (that has got output port reservation) is transferred from the random-access buffer to the send buffer in that IB module. The cells stored in a send buffer are shifted forward by one position, in every time slot, and the cell in the HOL position will be transported across the switch fabric. A scheduled cell is stored at such a location in the send buffer that, precisely at the time slot for which it has got the reservation of the desired output (i.e., at the scheduled transmission time of that cell), it will reach the HOL position of the send buffer and will be transported across the switch fabric.

2.1. Reservation Procedure

First, we define some terminologies. By 'time t' in our description, we mean time slot t.

Definition 1 *Reservation time of an input port i at time t is defined as the time at which the cell, for which the reservation is made at time t, is actually transmitted across the switch fabric, and is denoted by $R_i(t)$.*

Definition 2 *Reservation interval of an input port i at time t is $R_i(t) - t$.*

The reservation interval (RI) of each input port determines the reservation priority of that input port. This point will become clear later, with the description of the reservation scheme.

Definition 3 *The first reservation input port (FRP) at a given time t is the input port i with the largest reservation interval $R_i(t)$ among all the input ports ($i \in \{0, 1, \ldots, N-1\}$).*

Definition 4 *The last reservation input port (LRP) at a given time t is the input port i with the smallest reservation interval $R_i(t)$ among all the input ports ($i \in \{0, 1, \ldots, N-1\}$).*

In order to provide fairness, the reservation interval of each input port is allowed to change from time to time in a cyclic manner and hence it is called *cyclic reservation interval (CRI)*. For specific reason, which will be explained next, the cyclic reservation interval of input port i at time t is given by the relation

$$CRI_i(t) = \left\{ \begin{array}{ll} i & \text{if } t = 0 \\ (CRI_i(t-1) + \tau) \bmod N & \text{if } t \neq 0 \end{array} \right\} \quad (1)$$

where

$$0 \leq i \leq N-1,$$
$$GCD(\tau + 1, N) = 1, \quad \tau \neq qN \quad \text{for any } q \in I$$

The CR module of the switch has N reservation tables (RTs) in it, one RT for each IB [8]. Each RT consists of N reservation state entries to indicate the reservation states (i.e., the availability or nonavailability) of N output ports for a future time slot; a '0' indicates that the corresponding output port is available and a '1' indicates that it is not available. Output port address of the cells in the random-access buffer of IB_i, from the HOL position upto a maximum of d (called 'search depth'), are matched against the available output ports in RT_i. If a match is found, the corresponding entry of RT_i is changed from 0 to 1 and the matching cell is transferred from the random-access buffer to the send buffer in IB_i. In the send buffer, the cell is stored at the register entry corresponding to the value of $CRI_i(t)$, so that, after $CRI_i(t)$ number of time slots, it will reach the HOL position of the send buffer and will be transported across the switch fabric. In the random-access buffer, all the subsequent cells (those were behind the cell which has just been transferred to the send buffer) are shifted forward to occupy the vacant position. Schematic representation of the structure and operation at IB_i-RT_i pair is given in Fig. 2.

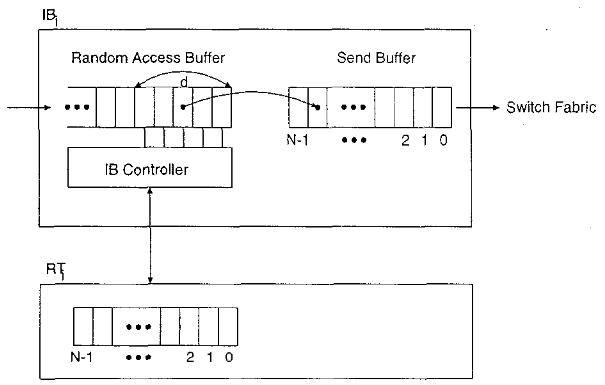

Figure 2. Schematic representation of the structure and operation at IB_i-RT_i pair

With the above reservation mechanism, additional hardware is necessary for the IB module than proposed in [8], in order to maintain cell sequence within a flow. Since the reservation interval of an input port changes cyclically from time to time, it is possible in our scheme that, two cells, from the same input port and with the same destination address, to have the assigned transmission times in an order that reverses the order in which they arrived at the IB. An exchange of the storage positions of these cells within the send buffer is necessary. This can be done with minimal hardware modification. The problem of maintaining cell sequence within a flow using extra hardware has been reported in other schedulers also [9, 7].

2.2. Parallelism

We have mentioned in the description of the reservation procedure that, in each time slot, output port address of the cells in the random-access buffer of IB_i, from the HOL position upto a maximum of d, are matched against the available output ports in RT_i. All cells within the search depth d are matched simultaneously. However, only the oldest cell with a matching is selected and transferred to the send buffer. The hardware implementation details of such an IB controller is presented in [8]. The advantage of the parallel searching scheme is that the scheduling speed is invariant to the value of d; a large value of d has great impact on the performance of the switch as shown in the next section. Ofcourse, d cannot be too large as it increases the hardware complexity of the IB controller. The concept of sending many requests simultaneously by an input port, in a single time slot, has been used in several schedulers [3, 5, 6, 9, 7].

2.3. Pipelining Technique

In each time slot, reservation mechanism proceeds at each IB-RT pair independently and in a pipelined manner. Reservation state entries which are modified as described in the reservation procedure are cyclically shifted from one RT to another. In each time slot t, all entries in the RT associated with the FRP at time t are set to 0 before the reservation mechanism for that IB-RT pair takes place; similarly, entries in the RT associated with the LRP at time t are discarded after the reservation mechanism for that IB-RT pair has been finished.

The sequence in which this cyclic shifting occurs is different for different values of τ and N. For a given τ and for a fixed N, we can obtain the relationship between i and j, where the entries of RT_j are shifted to RT_i, in the manner described next.

Since input port j (resp. i) makes a reservation at time t (resp. $t+1$) for the actual cell transmission at $R_j(t)$ (resp. $R_i(t+1)$), if there is a matching in the respective case, we have

$$\begin{aligned} R_j(t) &= R_i(t+1) \\ t + CRI_j(t) &= t + 1 + CRI_i(t+1) \\ t + ((CRI_j(0) + \tau t) \bmod N) &= t + 1 + ((CRI_i(0) + \tau(t+1)) \bmod N) \\ j + \tau t + kN &= (i + 1 + \tau(t+1)) \quad \text{for some } k \in I \\ j + kN &= (i + 1 + \tau) \\ j &= (i + 1 + \tau) \bmod N \qquad (2) \end{aligned}$$

Step 2 follows from Def. 2 and steps 3 and 4 follow from Eqn. (1).

Observation 1 *The sequence satisfying Eqn. (2) includes all the N RTs, provided $GCD(\tau+1, N) = 1$.*

Proof. Using Eqn. (2), we get the N successive RT indices as

$$i,\ (i+\tau+1) \bmod N,\ (i+2(\tau+1)) \bmod N,\ \ldots, (i+(N-1)(\tau+1)) \bmod N \qquad (3)$$

If these N indices are not all different, then for some $0 \leq j, k \leq N-1, j \neq k$

$$(i + j(\tau+1)) \bmod N = (i + k(\tau+1)) \bmod N$$

i.e.

$$(j(\tau+1)) \bmod N = (k(\tau+1)) \bmod N$$

By the *cancellation property* [10], when $GCD(\tau+1, N) = 1$, we have

$$j \bmod N = k \bmod N$$

which contradicts the assumption that $j \neq k$. Therefore, all the N RT indices in the sequence given by Eqn. (3) are distinct. ■

2.4. Cyclic Reservation Interval Algorithm

Here, we summarise the sequence of operations which is performed at IB_i-RT_i pair, at time t. The same sequence of operations is performed at all IB-RT pairs simultaneously, in every time slot.

1. The cyclic reservation interval of input port i, $CRI_i(t)$, is calculated according to Eqn. (1).
2. If $CRI_i(t) = N-1$, all the entries of RT_i are set to 0 ($CRI_i(t) = N-1$ means that the input port i is the FRP at time t).
3. Output port address of up to a maximum of d cells from the HOL position, in the random-access buffer of IB_i, are matched against the vacant output port reservation state entries in RT_i. If a matching is found, then the corresponding entry of RT_i is made 1.
4. If $CRI_i(t) = 0$, the entries of RT_i are discarded ($CRI_i(t) = 0$ means that the input port i is the LRP at time t). Otherwise, the entries of RT_i are shifted to its successor RT. The entries of RT_i are replaced by those of its predecessor.
5. The matching cell (if any) is transferred from the random-access buffer to the send buffer in IB_i, and is stored at the location corresponding to the value of $CRI_i(t)$. If there is a cell with the same output port address behind this location, then an exchange of the storage positions of these cells within the send buffer is done, in order to maintain cell sequence within that flow.
6. If a cell is transferred to the send buffer, then all the subsequent cells in the random-access buffer are shifted forward to occupy the vacant position.

Next, we define two properties of the cyclic reservation interval scheduler which ensure the correctness of the distributed algorithm, and guarantee efficiency.

Property 1 *At any given time t, no two input ports i and j ($i \neq j$) have the same reservation time and hence there is no 'output reservation conflict'.*

Property 2 *In any time slot, no input port is denied a reservation (if any one of the required output port is available).*

The proofs of the above properties are given elsewhere.

3. Performance

In this section, we use simulation to study in detail the performance of the switch with the cyclic reservation interval scheduler. We obtain the performance measures such as maximum achievable throughput of the switch, mean cell delay, and cell loss ratio. In getting the cell delay performance, we assume large random-access buffers in the IB modules. In this paper, we present the results with independent and uniform traffic, i.e., at each input, at the beginning of each time slot, a cell arrives with a fixed probability and the arriving cell has a destination request that is uniformly distributed over all outputs, independent of other cells.

3.1. Simulation Results

Table 1 shows the maximum achievable throughput for various switch sizes and search depths (N and d resp.). In the throughput calculation, we assume that the input queues are 'saturated', i.e., each input queue as always having d or more number of cells in it. Note that the rate of increase of throughput with search depth is more and more as switch size changes from 4 to 8 and to 16. Also, with a reasonable value of search depth of 16, maximum throughput is as high as 96%.

d	$N=4$	$N=8$	$N=16$
1	0.672	0.636	0.617
2	0.779	0.753	0.741
3	0.831	0.811	0.803
4	0.864	0.847	0.841
5	0.886	0.871	0.868
6	0.902	0.889	0.888
7	0.914	0.903	0.902
8	0.923	0.915	0.914
9	0.931	0.924	0.924
10	0.937	0.932	0.932
12	0.946	0.945	0.944
14	0.954	0.954	0.954
16	0.96	0.961	0.961

Table 1. Maximum achievable throughput for various switch sizes and search depths (N and d resp.)

Fig. 3 shows the mean delay of a cell (expressed in number of cell times) in an input queueing switch with the proposed scheduler, and also in an output queueing switch, for comparison. We ignored the unit cell transmission time across the switch fabric. Also, the scheduler is so fast that the processing delay is negligible. Thus, the cell delay has two components: waiting time in the random-access buffer before the transmission time being assigned, and the waiting time in the send buffer after the transmission time being assigned. Note that the second component is the inherent delay due to the advance reservation. This dealy is independent of the cell arrival rate and its mean value is equal to $(N-1)/2$, where N is the number of input ports. Ofcourse, except for small size switches, this delay component may be offensive. We discuss later, some applications where the proposed scheduler is most appropriate.

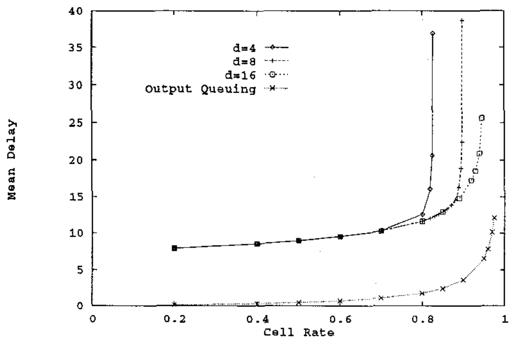

Figure 3. Mean cell delay (in cell times) vs. cell arrival rate for various search depths d; N =16

Figs. 4 and 5 illustrate the cell loss behaviour. Offered load is the same as the cell arrival rate. The results corresponding to smaller buffer sizes than the search depths (e.g., buffer size 5 and search depth 8) mean that all the cells present in the input random-access buffer are matched in the same slot, against the reservation state entries in the RT. Our simulation results show that it is adequate to provide a buffer size of 20 cells, as there is no further improvement beyond this point.

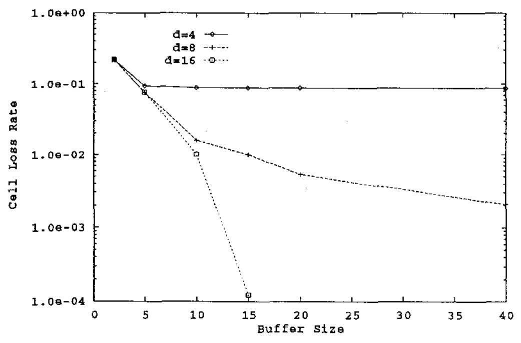

Figure 4. Cell loss rate vs. random-access buffer size for various search depths d; offered load is 0.9 and N =16

3.2. Discussion

One drawback with our scheduler is that when the offered load is small, there is a mean cell delay (in cell times) equal

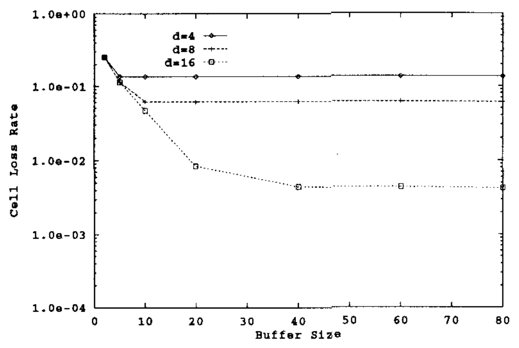

Figure 5. Cell loss rate vs. random-access buffer size for various search depths d; offered load is 0.95 and N =16

to $(N-1)/2$, where N is the number of the input ports. As mentioned earlier, this is the inherent delay of the advance reservation scheme and is the mean waiting time of the cells after reservation but before being transferred across the switch fabric, and also it is independent of load. Due to this delay performance, our scheduler is most appropriate for switches of small to moderate size with high link speeds (1 Gbit/s or more), such as those used in ATM LANs (local-area networks). For example, this delay is about 2.5 μs if the number of input ports is 16, and link speed is 1.3 Gbit/s. Large switches are often inappropriate for LANs as it would be unduly costly for sites that have only a few workstations. Smaller switches allow capacity to be added incrementally at low cost [3]. A number of work stations can be connected via a multiplexer to a high speed link. Our scheduler is also appropriate for large switches with a modular architecture, where switch modules are independently operated and each switch module has substantially fewer inputs than outputs.

Also, this delay performance is not a serious drawback even in a traffic scenario where we have real time voice or video traffic in addition to the data traffic. Since this delay component is load independent and its maximum value is known (i.e., $N-1$), we can have a 'built-in' playout delay at the receiving end of the real time traffic.

4. Conclusion

We have proposed a new output scheduler to improve the performance of an input queueing ATM switch. Without any speed-up, input/output grouping or complicated hardware, the maximum throughput achievable for a moderate search depth of 16 is about 96%. As far as the hardware complexity is concerned, the new scheme is comparable with the one proposed in [8]. However, the performance of the new scheme is much better than that reported there. For example, the maximum achievable throughput there, even for an arbitrarily large (impractical) search depth, is only 94%. Our simulation results reveal superior performance in cell delay and cell loss ratio also. Furthermore, the new scheme has got the inherent fairness property without any additional hardware complexity. The high speed and the efficiency of the new scheme make it most appropriate for switches with small size and high speed input/output links, such as those used in ATM LANs, where the traffic from a number of workstations are multiplexed on to high speed links to a central ATM switch.

References

[1] M. G. Hluchyi and M. J. Karol, "Queueing in High-Performance Packet Switching," *IEEE JSAC,* 6(9):1587-1597, 1988.

[2] L. Jacob and A. Kumar, " Saturation Throughput Analysis of An Input Queueing ATM Switch with Multiclass Bursty Traffic," *IEEE Tr. on Communications,* 43(2/3/4):757-761, 1995.

[3] T. E. Anderson, S. S. Owicki, J. B. Saxe, and C. P. Thacker, "High-Speed Switch Scheduling for Local-Area Networks," *ACM Tr. on Computer Systems,* 11(4):319-352, 1993.

[4] N. McKeown, P. Varaiya, and J. Walrand, "Scheduling Cells in an Input-Queued Switch," *IEE Electronics Letters*, 29(12):2174-2175, 1993.

[5] H. Obara, "Optimum Architecture for Input Queueing ATM Switches," *IEE Electronics Letters*, 27(3):555-557, 1991.

[6] H. Obara, S. Okamoto, and Y. Hamazumi, "Input and Output Queueing ATM Switch Architecture with Spatial and Temporal Slot Reservation Control," *IEE Electronics Letters*, 28(1):22-24, 1992.

[7] M. J. Karol, K. Y. Eng, and H. Obara, "Improving the Performance of Input Queued ATM Packet Switches," *INFOCOM*:110-115, 1992.

[8] H. Matsunaga and H. Uematsu, "A 1.5 Gb/s 8x8 Cross-Connect Switch Using a Time Reservation Algorithm," *IEEE JSAC* 9(8):1308-1317, 1991.

[9] H. Obara and Y. Hamazumi, "Parallel Contention Resolution Control for Input Queueing ATM Switches," *IEE Electronics Letters*, 28(4):838-839, 1992.

[10] R. Graham, D. Knuth and O. Patashnik, "Concrete Mathematics: A Foundation for Computer Science," *Addison Wesley,* 1988.