

Cell-Level Path Allocation in a Three-Stage ATM Switch

Martin Collier and Tommy Curran,

School of Electronic Engineering, Dublin City University
Dublin 9, Ireland.

collierm@eeng.dcu.ie, currant@eeng.dcu.ie

ABSTRACT A method of cell-level path allocation for three-stage ATM switches has recently been proposed by the authors. The performance of ATM switches using this path allocation algorithm has been evaluated by simulation, and is described here. Both uniform and non-uniform models of output loading are considered. The algorithm requires knowledge of the number of cells requesting each output module from a given input module. A fast method for counting the number of requests is described.

1. Introduction

A new method of allocating paths through a three-stage ATM switch has recently been proposed by the authors [1]. The method applies to a switch featuring intermediate channel grouping, such as that in Fig. 1. The algorithm hunts sequentially for available paths through the intermediate stage of the switch, but multiple searches are conducted in parallel, so that comparatively few iterations are required to search through all possible paths. Hence cell-level path allocation can be performed, i.e., the updating of paths after every time slot. A full description of the algorithm, and its implementation, may be found in [1]. It uses an array of $L_1.L_2$ processors called *atomic*() processors to achieve the necessary parallelism (where L_1 and L_2 are as defined in Fig. 1). Each processor must be initialised with the value of K_{ij} , the number of cells from input module i requesting output module j .

The method for counting cell requests described in [1] has an execution time which increases in proportion as the switch size increases. This can limit the maximum achievable switch size. A faster method of obtaining the value of K_{ij} is described in Section 2.

The performance of switches using this path allocation algorithm is considered in Section 3 (for uniform loading of outputs) and Section 4 (for non-uniform loading).

2. A faster method of request counting

The method of request counting described in [1] has the disadvantage that it *sequentially* determines the number of cells requesting output modules L_2-1 , L_2-2 , etc. The total number of clock cycles required is $n_1 + L_2$, i.e., the sum of the number of input ports per input module, and the number of output modules. Hence, the required clock rate may be excessive in a large switch, given that request counting, path allocation, and routing tag assignment, must (ideally) all take place within the duration of one time slot.

A faster, parallel, implementation would simultaneously calculate K_{ij} (the number of requests from input module i for output module j) for all values of j ($0 \leq j < L_2$). Suitable hardware will now be described. The execution time for this hardware is $2\lceil \log_2(n_1 + L_2) \rceil + 1$ clock cycles, which is substantially less than that for the sequential hardware described in [1], at the price of an increase in hardware complexity.

The hardware required is shown in Fig. 2. Data cells from the n_1 input ports associated with input module i are merged with L_2 control packets (one per output module) by a Batcher sorting network. The merge operation is performed in such a way that idle cells (i.e., empty cells from inactive input ports) are sorted to the highest-numbered output ports of the Batcher network. If the control packet for output module j appears at output D_j of the Batcher network, then the data cells (if any) requesting that output module appear at lower-numbered output ports of the sorter (ports D_{j-1} , D_{j-2} , etc.), as shown in Fig. 2.

Under these circumstances, it may readily be shown that

$$D_j = \sum_{u=0}^j K_{iu} + j$$

where K_{iu} is the number of data cells requesting output module u , and i is fixed, since the Batcher network processes only requests from input module i .

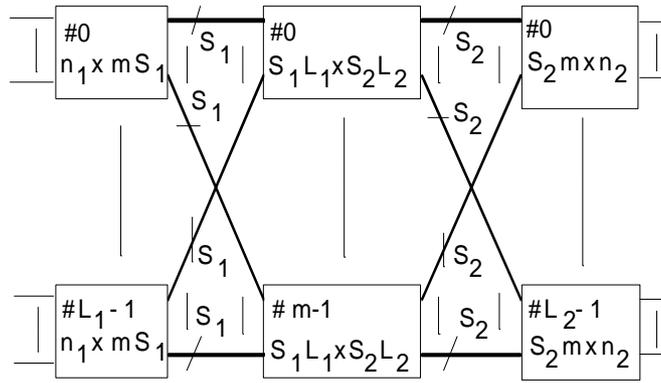
The key to the new method of request counting is the observation that

$$K_{ij} = \begin{cases} D_0, & j = 0 \\ D_j - D_{j-1} - 1, & j > 0 \end{cases}$$

The necessary subtraction can be performed very efficiently, since

$$D_j - D_{j-1} - 1 = D_j + \bar{D}_{j+1}$$

where \bar{D} is the 1's complement of D , obtained by bitwise inversion of D . It follows that the value of K_{ij} can be generated using a serial adder, and can then be stored in the K register of the appropriate *atomic*() processor (labelled X_{ij} in [1]).



S_1 (S_2): the number of channels in the channel group connecting each input (output) module to each intermediate switch module.
 V : channel rate (155 Mb/s)
 n_1 (n_2): the number of input (output) ports per input (output) module;
 m : the number of intermediate switch modules;
 L_1 (L_2): the number of input (output) modules;

Fig. 1 A three-stage switch with intermediate channel grouping

It is necessary to generate a concentrated list of the values $D_0, D_1, D_2, \dots, D_{L_2-1}$ as input data for the serial adders. These values are available at the address generators which have received control packets from the sorter outputs, but are not concentrated onto contiguous outputs. Hence a concentrator is required. This is the purpose of the binary self-routing network shown in Fig. 2, which is variously known as the indirect binary n-cube [2] and the 'reverse banyan' [3]. The address generators forward only control packets to this network. Address generators which have received a data cell or an idle cell through the Batcher network submit an inactive packet to the concentrator. The address generator which receives control packet j from output D_j of the Batcher network appends a data field to the packet containing the value of D_j . This packet is then routed to output j of the concentrator. A total of L_2 control packets is thus simultaneously launched into the concentrator, and these are routed to the serial adders at outputs zero through L_2-1 , without blocking. The absence of blocking within the concentrator will be verified in Appendix A.

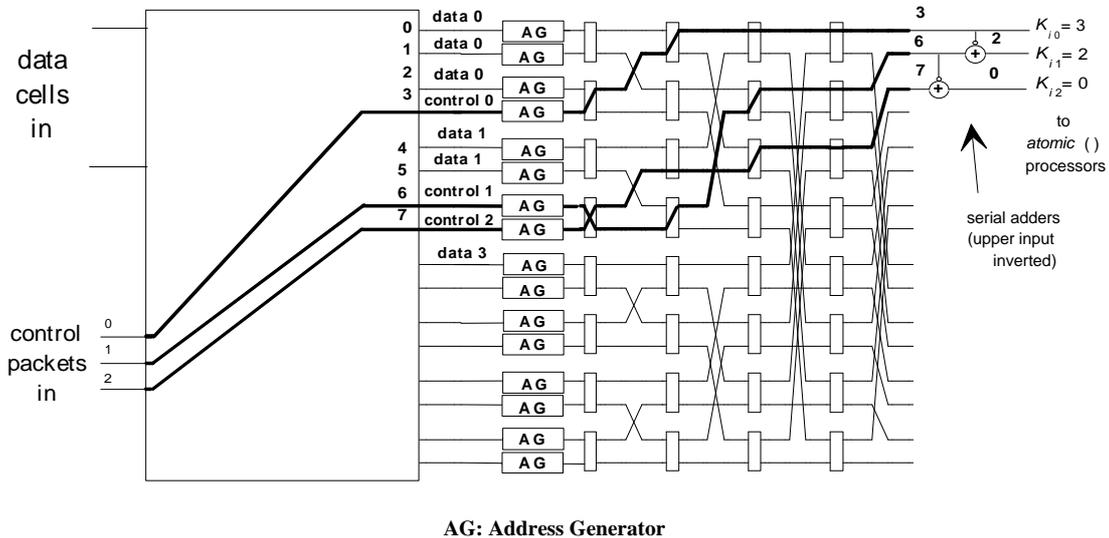


Fig. 2 A faster method of request counting.

The concentrated list of D_j values is then read by these serial adders, the upper input (as shown in Fig. 2) being inverted. Hence the K_{ij} values are generated, and passed to the *atomic()* processors. The example considered shows three requests for output module zero, two for output module one, and none for output module two. It can be seen that the correct values (i.e., 3, 2 and 0) are returned to processors X_{i0} , X_{i1} and X_{i2} respectively. The submitted packets take two cycles to propagate through each stage of the concentrator (one cycle to identify if the packet is active, and another to determine where to route it) and an additional clock cycle is required before the serial adder generates the least significant bit of the appropriate K_{ij} value. Hence the number of clock cycles required by the request count hardware before path allocation can commence is

$$2\lceil \log_2(n_1 + L_2) \rceil + 1.$$

This is much less than the $n_1 + L_2$ cycles required by the hardware described in [1]. Taking as an example the sample switch considered in that paper, for which $n_1 = 96$ and $L_2 = 32$, the number of clock cycles is reduced from 128 to 15.

3. Performance with uniformly loaded output modules

The performance of a three-stage switch using the cell-level path allocation algorithm described above will now be evaluated. The cell loss probability must be determined by simulation since no analytical method is currently available. The simulation model is based on the following assumptions:

- (i) There is no input queueing; cells which are not allocated a path on the first attempt are discarded.
- (ii) The switch is offered a maximum load; each input port of the switch submits a cell in every time slot.
- (iii) The destination of each cell is drawn from a uniform distribution; all output modules receive the same load.
- (iv) The switch modelled is that shown in Fig. 1, for various choices of the parameters L_1, L_2, m, S_1, S_2 and n_1 .
- (v) The maximum number of cells generated is 10^9 . If zero cell loss is recorded during the simulation, the cell loss probability is assigned the value 5×10^{-11} . The probability of losing contention is assumed to be independent for each cell, at low levels of loss. With this assumption, the probability that the cell loss probability (CLP) is below 5×10^{-11} , given that no losses were recorded, is above 95%, i.e.,

$$\Pr[CLP < 5 \times 10^{-11}] = (1 - 5 \times 10^{-11})^{10^9} > 0.95.$$

- (vi) The cell loss probability is assumed to be equal to the expected number of cells lost per time slot, as a proportion of the offered load.

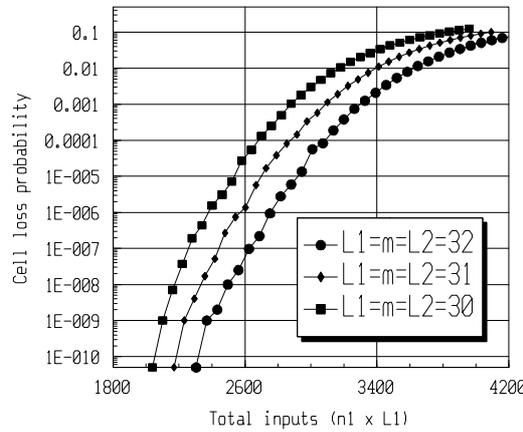


Fig. 3 Performance with uniform traffic
($L_1 = m = L_2, S_1 = S_2 = 4$)

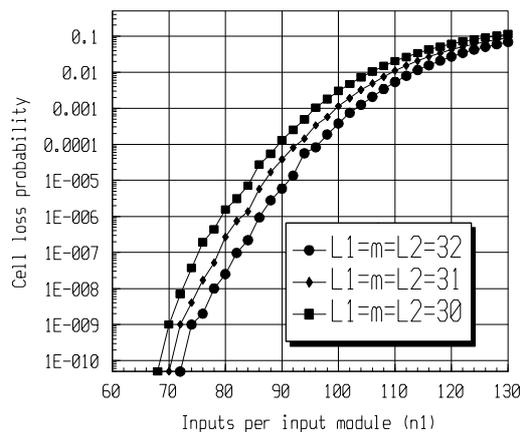


Fig. 4 Variation of cell loss with n_1
($L_1 = m = L_2, S_1 = S_2 = 4$).

The influence of the choice of channel group size in Fig. 1 on the cell loss probability is considered in Figs. 3-6. The results where $S_1 = S_2 = 4$ are shown in Fig. 3. Note that $L_1 = L_2 = m$ for the three switches simulated. The curves obtained show that, with the number of switch inputs fixed, the cell loss probability falls as the number of switch modules is increased, as expected. A more interesting result is observed if the results are plotted for a fixed size of input module, as shown in Fig. 4. These loss curves indicate that, if modular growth is achieved by increasing L_1, L_2 and m in the same proportion, the cell

loss probability will decrease. Hence, the additional loss due to the higher switch load is offset by the increasing diversity of paths available.

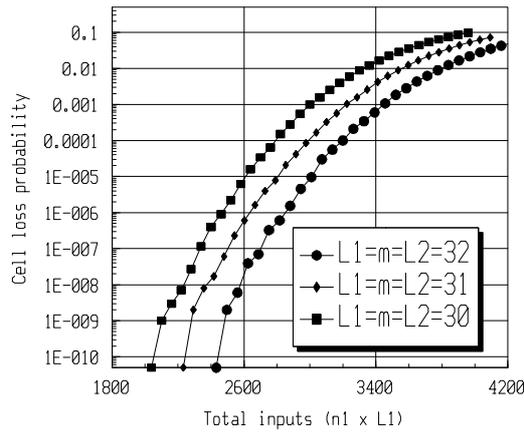


Fig. 5 Performance with uniform traffic ($L_1 = m = L_2, S_1 = 8, S_2 = 4$).

Fig. 5 shows the corresponding results for three similar switches, where S_1 equals 8. Doubling the channel group size at the input side of the intermediate stage gave rise to only a marginal decrease in the cell loss probability. Doubling the channel group size at the output side of the intermediate stage reduces the loss considerably, as shown in Fig. 6.

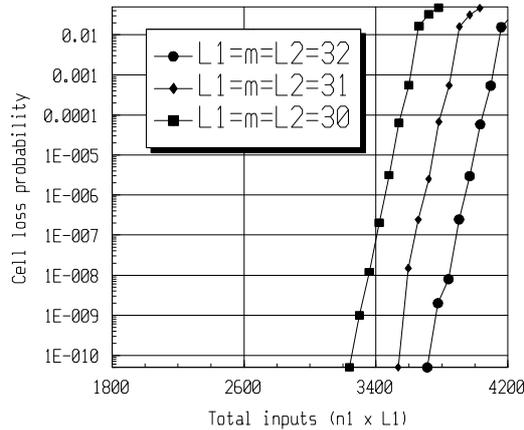


Fig. 6 Performance with uniform traffic ($L_1 = m = L_2, S_1 = 4, S_2 = 8$).

These simulations indicate that the intermediate modules should be designed as expansion modules, with more outputs than inputs, to obtain the best performance. This seems intuitively reasonable; a cell may be routed to any intermediate module, but can be routed to only one output module. An alternative to increasing S_2 is to change the value of m . However, this has the disadvantage that the number of iterations required by the path allocation algorithm will increase, so that higher speed hardware may be required. The effect of varying m in the range 30 to 34, for a switch with $L_1 = L_2 = 32, S_1 = 4$ and $S_2 = 8$ is shown in Fig. 7.

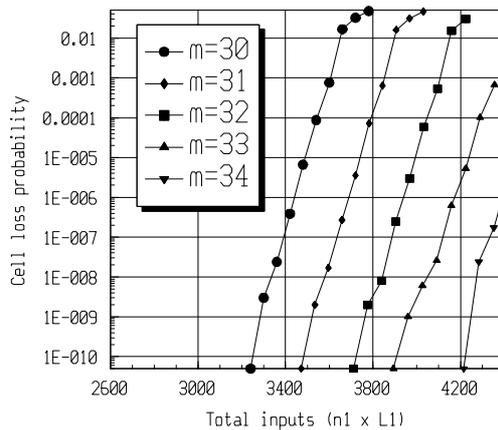
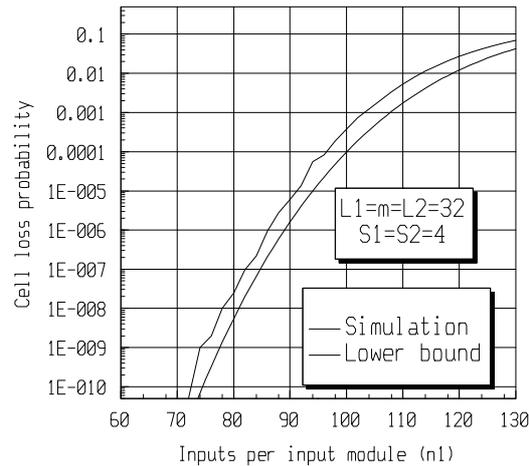


Fig. 7 Effect of changing m ($L_1 = L_2 = 32$, $S_1 = 4$, $S_2 = 8$).

Fig. 8 Lower bound compared to simulation results ($L_1 = m = L_2 = 32$, $S_1 = S_2 = 4$).

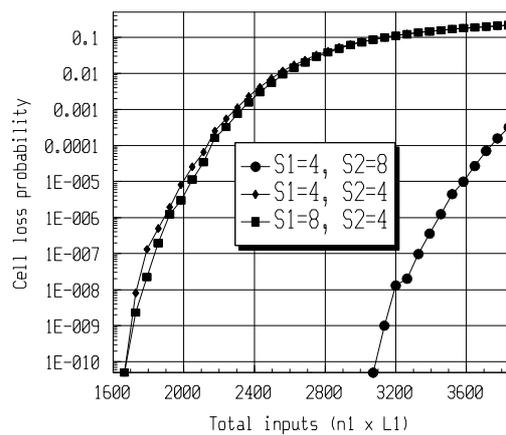
The efficiency of the path allocation algorithm may be demonstrated by comparing its performance with the lower bound on achievable performance. This bound corresponds to the knockout loss in [4]; at most $S_2 m$ cells can be delivered to each output module in any one time slot. This bound, for the case where $L_1 = L_2 = m = 32$ and $S_1 = S_2 = 4$, is compared to the simulation results in Fig. 8. It can be seen that the performance of the algorithm is close to the lower bound.

4. Performance with non-uniformly loaded output modules

The above results all apply to a situation where there is a uniform load across all the output modules. Modifying assumption (iii) of the simulation model allows non-uniform loads to be assessed.

Fig. 9 shows the results obtained for a switch with $L_1 = L_2 = m = 32$, for various channel group sizes, in the case where 75% of arriving cells request one from sixteen (contiguous) output modules, each output module of the sixteen being selected with equal probability. The remaining 25% of cells select (with equal probability) one of the other sixteen output modules. Hence 75% of the load is offered to only 50% of the available outputs.

It can be seen that the cell loss probability increases dramatically, compared with the case of uniform loading (i.e., Fig. 3 and Fig. 5), when $S_1 = S_2 = 4$, or when $S_1 = 8$ and $S_2 = 4$. Furthermore, the additional bandwidth available from the input stage of the switch to the intermediate stage in the latter case results in a negligible decrease in the cell loss probability. In contrast, the cell loss probability increases by a relatively small amount (compared with Fig. 6) for the case where $S_1 = 4$ and $S_2 = 8$. This further underlines the benefit of having a large bandwidth at the output side of the intermediate stage.


Fig. 9 Performance with nonuniform traffic for various values of intermediate stage bandwidth ($L_1 = m = L_2 = 32$).

The effect of progressively increasing the asymmetry of the load on the switch outputs shall now be investigated in the case of a switch where $L_1 = L_2 = m = 32$, $S_1 = 4$, $S_2 = 8$ and $n_1 = 96$.

The output modules are divided into two groups. One group (termed the demand group) contains k output modules, with $0 < k < L_2$. The probability of a cell requesting an output module in the demand group is chosen so that the expected number of cells requesting each module in the demand group is $r \cdot S_2 m$, where $0 < r < 1$. If a cell does not request an output module in the demand group, it requests one of the remaining $L_2 - k$ output modules, with uniform probability.

The cell loss probability is shown in Fig. 10 for various values of k and r , in the case where the output modules in the demand group are contiguous. The probability of cell loss is shown only for cells requesting an output module in the demand group. No losses were recorded in the simulations for cells requesting the background group.

The probability of loss can be seen to increase significantly as the size of the demand group (k) is increased. However, this probability stays below 10^{-10} if the load offered to each output module is below 55% of its input capacity (i.e., if $r < 0.55$). Since each output module has 256 inputs, each can deliver data at the full ATM rate to as many as 140 output ports, without excessive cell loss, even in the presence of a severe traffic imbalance. Setting $n_2 = 140$ in Fig. 1, with the values of the other switch parameters as chosen above, results in a 3072×4480 switch, with a cell loss probability below 10^{-10} in the presence of a 100% offered load and an asymmetric loading of the outputs. A square (3072×3072) switch should have a much lower figure for cell loss.

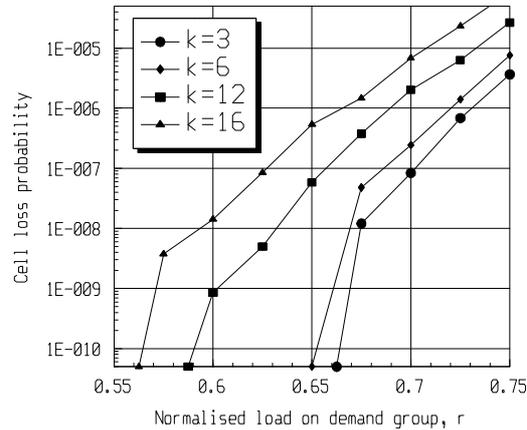


Fig. 10 Variation of cell loss with traffic imbalance, where contiguous output modules have a high load ($L_1 = m = L_2 = 32, S_1 = 4, S_2 = 8, n_1 = 96$).

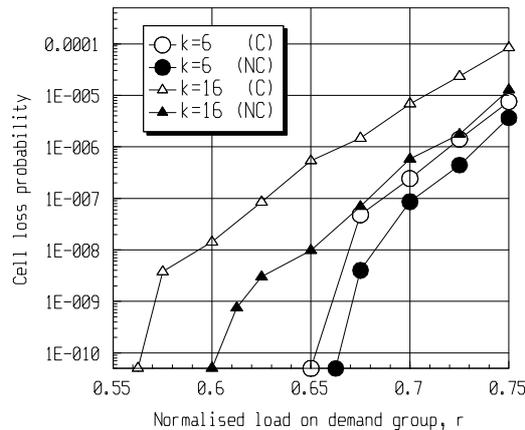


Fig. 11 Comparison of performance for contiguous and non-contiguous demand groups ($L_1 = m = L_2 = 32, S_1 = 4, S_2 = 8, n_1 = 96$).

The cell loss probability depends to some extent on the pattern of traffic imbalance present. Fig. 11 compares the loss for a switch with a contiguous demand group and a switch with a demand group whose output modules are interspersed with those carrying background traffic. The variation in cell loss probability is most pronounced in the case of the more unbalanced load ($k = 16$). Note that the cell loss probability is higher for the contiguous demand group.

5. Concluding remarks

The parallel request counting algorithm described here allows the necessary clock rate for the path allocation algorithm described in [1] to be reduced and/or a larger switch to be constructed, compared with the original method of request counting.

The effect of varying the parameters of the switch in Fig. 1 on its performance has been determined by simulation. The use of expansion switch modules in the intermediate stage of the switch has been suggested as an alternative to increasing the number of intermediate modules, as a means of reducing cell loss.

It has been demonstrated that the algorithm is not unduly sensitive to imbalances in the offered load. Only the case of a 100% offered load has been considered here; the case where the switch inputs are operating well below saturation requires further investigation.

Appendix A

A sufficient condition for the reverse banyan in Fig. 2 to be non-blocking is that, for any pair of input ports I_1 and I_2 , requesting output ports O_1 and O_2 respectively, with $I_2 > I_1$ and $O_2 > O_1$, the following is true:

$$O_2 - O_1 \leq I_2 - I_1.$$

This (well-known) result may be established, for example, by appropriately modifying the proof in Appendix A of [5], which pertains to a (forward) banyan. In the present application, the input ports are

$$I_1 = D_j$$

and

$$I_2 = D_k$$

and the output ports are

$$O_1 = j$$

and

$$O_2 = k,$$

with $k > j$.

Hence the non-blocking condition becomes

$$k - j \leq D_k - D_j.$$

The number of requests for each output module must be non-negative, i.e.,

$$K_{ij} \geq 0.$$

Since

$$D_k - D_j = \sum_{u=j+1}^k K_{iu} + k - j$$

it follows that the condition for the network to be non-blocking is always satisfied.

References

- [1] M. Collier and T. Curran, "Path allocation in a three-stage ATM switch with intermediate channel grouping", Proc. INFOCOM '93, Session 8b, pp. 927-934, April 1, 1993.
- [2] M.C. Pease, "The indirect binary n-cube microprocessor array", IEEE Trans. Comput., vol. 26, no. 5, pp. 250-265, May 1977.
- [3] H. Kim and A. Leon-Garcia, "A multistage ATM switch with interstage buffers", Proc. of the International Switching Symposium, Stockholm, 1990, vol. V, pp. 15-20.
- [4] K. Eng et al., "A modular broadband (ATM) switch architecture with optimum performance", Proc. of the International Switching Symposium, Stockholm, 1990, vol. IV, pp. 1-6.
- [5] T.T. Lee, "Nonblocking copy networks for multicast packet switching", Journal Of Select. Areas Commun., Vol. 6, no. 9, pp. 1455-1467, Dec. 1988.