A Fully Desynchronized Round-Robin Matching Scheduler for a VOQ Packet Switch Architecture

Ying Jiang and Mounir Hamdi
Department of Computer Science
Hong Kong University of Science and Technology
Clear Water Bay, Kowloon, Hong Kong
Email: hamdi@cs.ust.hk

Abstract- **Virtual Output Queuing (VOQ) is a practical and high-performance packet switch architecture. There are many simple iterative arbitration algorithms proposed for the VOQ architecture. We investigate in this paper the performance of various such algorithms and based on the analysis of pointer desynchronization effect, we propose a group of new arbitration algorithms, called SRR (static round robin matching) which perform pretty well under various traffic models and are easy to implement in hardware.**

I. Introduction

The traditional output queued (OQ) switch architecture is appreciated for its optimal performance, and is frequently used as a yardstick by which the performances of newly proposed switch architectures are measured. OQ switches can always achieve 100% throughput since up to N packets can be transferred to a single output port in a time slot for an N x N switch. However, the high internal speed up required, N, makes it impractical to build these switches for a large number of ports and/or for high line rates. In contrast, input queued (IQ) switches are designed to operate with a switching fabric running at an internal rate equal to the external link speed. Unfortunately, when using a first-in-first-out (FIFO) queuing discipline at the input queues, due to the head-of-the-line (HoL) blocking problem, they only provide a maximum throughput of 58.6% [1] under uniform traffic and much lower than that for other traffic models.

An architecture called virtual output queuing (VOQ) [2] is proposed to solve the HoL problem while achieving the scalability of IQ switches. Rather than maintaining a single FIFO queue for all cells, each input maintains a separate queue for the cells directed to different outputs. In this architecture, the switch performance essentially depends on its scheduling algorithm, that is, the arbitration between the input ports and the output ports. A good algorithm should achieve high performance, work with very high line rates and/or large number of input/output ports, and be simple to implement in hardware.

Many scheduling algorithms have appeared in literature. They can be classified into two types: approximating maximum size matching (MSM) and approximating maximum weight matching (MWM). Although the MWM algorithms have been proved to achieve 100% throughput under any traffic [3][4], they are too complex to be implemented and have a time complexity of $O(N^3 \log N)$. We instead concentrate in this paper on a group of more practical iterative algorithms, including RRM [5], iSlip [5], FIRM [7], etc., and evaluate their performances. Pointer desynchronization plays a very important role in these algorithms. Trying to get the best effect of desynchronization, we propose a set of *Static Round-Robin* (SRR) algorithms, which perform pretty well under various traffic models.

The rest of the paper is organized as follows: Section II introduces some background knowledge of iterative algorithms that approximate MSM. Section III gives the formal specifications of our algorithms. The simulation results are shown in Section IV. Section V gives a possible hardware design. Finally, Section VI concludes the paper.

II. Background Knowledge

A. Maximum Size Matching

The scheduling problem in an N x N switch is modeled using a bipartite graph (See Fig. 1), where each part contains N nodes, and one part corresponds to the input ports, while the second part to the output ports. The requests from the input queues to the corresponding output ports are represented as edges, creating a bipartite graph.

The maximum size matching (MSM) for a bipartite graph can be found by solving an equivalent network flow problem [10] and the algorithm is called *maxsize* here. The most efficient algorithm converges in $O(N^{5/2})$ time [9]. Although it is guaranteed to find a maximum match, it is too complex to implement in hardware and takes a long time to converge. Instead, simple iterative algorithms are often used to approximate MSM.

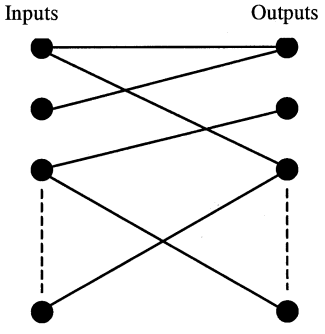

Fig. 1. Bipartite Graph

B. *Simple Iterative Algorithms: RRM, iSlip &FIRM*

There exists a group of algorithms that are easy to implement in hardware. They only take into consideration whether each VOQ is occupied or not, to make the scheduling decision and try to approximate MSM. The calculation of the matching is performed in an iterative fashion, where each iteration augments the matching calculated in the previous iteration. In each of the following iterations, only the unmatched inputs and outputs are considered. We will introduce three of these algorithms: RRM, iSlip and FIRM.

RRM (Basic Round-Robin Matching) [5] is the algorithm from which the well-known iSlip was developed. However, its performance is not good. The highest throughput under uniform traffic is no more than 65%. RRM works in the following way:

Step 1. Request. Each input sends a request to every output for which it has a queued cell.

Step 2. Grant. If an output receives any requests, it chooses the one that appears next in a fixed, round-robin schedule starting from the highest priority element. The output notifies each input whether or not its request was granted. The pointer to the highest priority element of the round-robin schedule is incremented (modulo *N*) to one location beyond the granted input. If no request is received, the pointer stays unchanged.

Step 3. Accept. If an input receives a grant, it accepts the one that appears next in a fixed round-robin schedule starting from the highest priority element. The pointer to the highest priority element of the round-robin schedule is incremented (modulo *N*) to one location beyond the accepted one. If no grant is received, the pointer stays unchanged.

iSlip was first described in [5]. The main characteristic of iSlip is its simplicity: it is readily implemented in hardware and can operate at high speed. The performance is also good. For uniform i.i.d. Bernoulli arrivals, iSlip is stable for any admissible load. It means that 100% throughput can be achieved. iSlip works in a similar way to RRM, with a small but important difference: in step 2 mentioned above, the pointer to the highest priority element only gets updated *if and only if* the grant is accepted by the input.

FIRM (Fcfs In Round Robin Matching) was proposed in [7]. The algorithm is almost the same as RRM and iSlip. We list in Table I, the difference among RRM, iSlip and FIRM in updating their pointers, which is the key difference among them. The updating scheme plays an important role in improving the performance. In [7], it is claimed that FIRM provides improvement over iSlip in average delay which reaches approximately 50% at loads above 95%.

C. *Pointer Desynchronization*

We can see that the difference of the 3 algorithms is only in updating their pointers, but they can have a huge difference in performance. We will explain below what really makes the difference is their abilities of desynchronizing their pointers.

We know that if several outputs grant the same input, no matter how this input chooses, only one match can be made, and the other outputs will be idle. To get as many matches as possible, it's better that each output grants a different input. Since each output will select the highest priority input if a request is received for it, it's better to keep the output pointers desynchronized. Fig. 2 shows the average number of synchronized output pointers under uniform traffic using iSlip, FIRM and RRM. We can see that when the load increases, the average number of synchronized pointers of iSlip decreases to be near 0, which means full desynchronization. While for RRM, this number changes little under light load and increases a lot under high load. For FIRM, it has an even better desynchronization effect than iSlip.

TABLE I

POINTER UPDATING SCHEMES OF RRM, ISLIP & FIRM

		RRM	iSlip	FIRM
Input	No grant	unchanged		
	Granted	one location beyond the accepted one		
Output	No request	unchanged		
	Grant accepted	one location beyond the granted one		
	Grant not accepted	one location beyond the granted one	unchanged	the granted one

It's not difficult to understand why RRM doesn't desynchronize pointers. If the initial values of the pointers are the same, and under heavy load, most pointers will advance synchronously, leading to many synchronized pointers. To understand why FIRM performs better than iSlip in terms of pointer desynchronization under heavy uniform traffic, the fact is that once any of the VOQ corresponding to a highest priority pointer is empty, iSlip will lose desynchronization, while FIRM still maintains it.

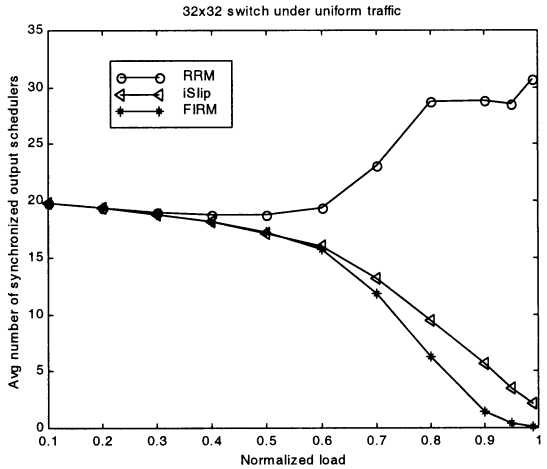

Fig. 2. Synchronization of output arbiters for the three algorithms for i.i.d Bernoulli arrivals with destinations uniformly distributed over all outputs

III. The Static Round-Robin Algorithms

In this section, we propose a group of new algorithms: Static Round-Robin (SRR) Matching. They are based on the idea of keeping full pointer-desynchronization.

From section 2.C we know that keeping pointers desynchronized is important. In fact, the best desynchronization is achieved if it is made artificially. That is, the pointers at the output side are set to be totally different at the beginning and advance synchronously at each time slot thereafter. A possible configuration for a 4 x 4 switch is shown in Table II.

This algorithm is called Single Static Round-Robin (SSRR). If we also consider the input side and force the input pointers to desynchronize, we will have Double Static Round-Robin (DSRR).

A. *Specifications of SSRR & DSRR*

The specification of SSRR is as follows:

Initialization. The input pointers are set to 0's. The output pointers are set to some initial pattern such that there is no duplication among the pointers.

The 3 steps of one iteration are:

Step 1. Request. Each input sends a request to every output for which it has a queued cell.

Step 2. Grant. If an output receives any requests, it chooses the one that appears next in a fixed, round-robin schedule starting from the highest priority element. The output notifies each input whether or not its request was granted. The pointer to the highest priority element of the round-robin schedule is always incremented by one (modulo *N*) whether there is a grant or not.

Step 3. Accept. If an input receives a grant, it accepts the one that appears next in a fixed round-robin schedule starting from the highest priority element. The pointer to the highest priority element of the round-robin schedule is incremented (modulo *N*) to one location beyond the accepted one.

For DSRR, in the initialization part, the input pointers will be set to the same pattern as the outputs. Step 3 is changed to:

Step 3. Accept. If an input receives a grant, it accepts the one that appears next in a fixed round-robin schedule starting from the highest priority element. The pointer to the highest priority element of the round-robin schedule is always incremented by one (modulo *N*) whether there is an accept or not.

TABLE II

POINTER CONFIGURATION FOR SRR

	Output 1	Output 2	Output 3	Output 4
Pointers at $4i^{th}$ time slot	4	3	2	1
Pointers at $(4i+1)^{th}$ time slot	1	4	3	2
Pointers at $(4i+2)^{th}$ time slot	2	1	4	3
Pointers at $(4i+3)^{th}$ time slot	3	2	1	4

B. *The Rotating Pointer Scheme*

There is a disadvantage of the algorithms described above: the inputs may be treated unfairly. To explain that, let's take a look at the unbalanced traffic model. The traffic model is as follows (for a 4x4 switch):

$$\begin{pmatrix} x & x & 0 & 0 \\ 0 & x & x & 0 \\ 0 & 0 & x & x \\ x & 0 & 0 & x \end{pmatrix}$$

The traffic is concentrated on two "diagonals". Consider output 1: no matter whether the highest priority input is 2, 3 or 4, it always gives the priority to input 4, while input 1 only gets the chance when the highest priority input is 1, or input 4 doesn't have any queued cell for output 1. So, under heavy load, input 1 and input 4 are treated very differently. To solve this problem, a scheme called "rotating pointers" is used as an enhancement of DSRR. In the step 2 of DSRR mentioned above, an output always searches from the highest priority one and increments by one each time (modulo *N*) until there is a request found. In particular, this is a clockwise rotation. Similarly, we can also have a counter-clockwise rotation, by searching in the other direction. If we use clockwise and counter-clockwise rotation alternatively, each for one time slot, we give each input the same chance to be served. For the input side, to get better performance, we always search in the counter-clockwise direction. We call this improved algorithm RDSRR (rotating DSRR).

IV. Simulation Results

The simulation results are gathered from a 32x32 switch. Delay is only the period of time a cell spends waiting in a VOQ before being transmitted. Each point in the figures runs for 500,000 time slots, and the statistics are gathered from the $50{,}000^{th}$ time slot. Average delay is calculated from all the cells outputted during this period of time. Relative average delay is the average delay divided by the average delay of an output-queued switch of the same size. By normalized load, we mean the percentage of time slots that have cells coming in, averaged over all inputs.

We evaluate the different algorithms under four traffic models: uniform, bursty, hotspot and unbalanced. For uniform traffic, the packets are Bernoulli arrivals, i.i.d., with destinations uniformly distributed over all outputs. For bursty traffic, busy and idle periods appear alternatively; in a busy period, there is a cell arriving in each time slot; in an idle period, there is no cell arriving in any time slot. The average loads for the inputs are the same, and the destinations are uniformly distributed burst by burst over all outputs. The traffic matrix of hotspot traffic is like (for a 4x4 switch):

$$\begin{pmatrix} 2x & x & x & x \\ 2x & x & x & x \\ 2x & x & x & x \\ 2x & x & x & x \end{pmatrix}$$

if output 1 is the "hotspot" and each flow is Bernoulli. It means that the "hotspot" has twice as much load as the other outputs. We've already introduced the unbalanced traffic in Section III.B.

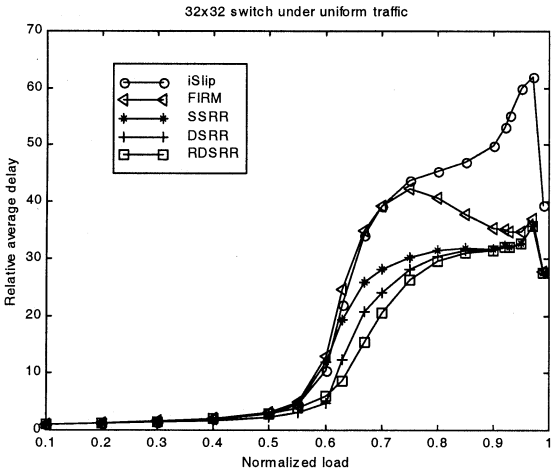

Fig. 3. Graph of average delay as a function of normalized load for a 32x32 switch under uniform traffic

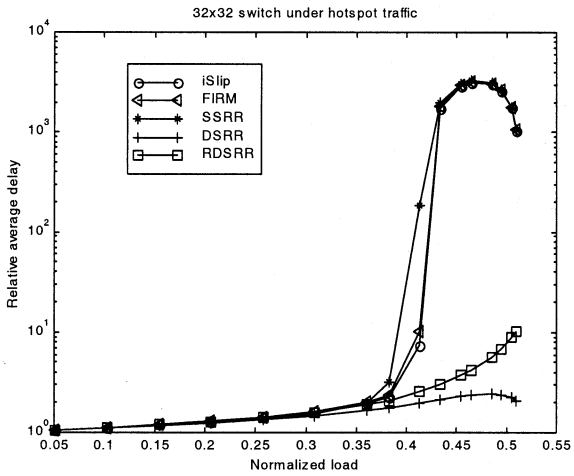

Fig. 5. Graph of average delay as a function of normalized load for a 32x32 switch hotspot traffic

In our simulation, we only consider admissible traffic, which means that no input or output is overloaded. The following figures show the results of one iteration.

Fig. 3 shows the results under uniform traffic. We can see that FIRM is better than iSlip under high load (over 0.7). SSRR, DSRR and RDSRR are all much better than FIRM and iSlip, especially when 0.5<load<0.95, and RDSRR has the best performance.

Fig. 4 shows the results under bursty traffic. The case is similar to the uniform case. But there is not much difference between iSlip and FIRM, and also between SSRR, DSRR and RDSRR.

Fig. 5 shows the results under hotspot traffic. iSlip and FIRM show similar performance. SSRR is a slightly worse than both them in a small range of load: 0.36-0.43, while DSRR and RDSRR are much better than the others.

The reason why DSRR and RDSRR perform so well is that by keeping the input pointers desynchronized, the "hotspot", output 1 always gets a chance to be served in each time slot.

Fig. 6 shows the results under unbalanced traffic. We can see that SSRR and DSRR don't perform well under this traffic model, while after using the rotating pointer scheme, the performance is much better than iSlip and FIRM.

V. Implementation of SRR Schemes

The implementation of SRR is really simple. The algorithms are almost the same as iSlip or FIRM, so we can directly use the architecture of iSlip (Fig. 7). A scheduling chip contains $2N$ arbiters: N grant arbiters to arbitrate on behalf of the outputs, and N accept arbiters to arbitrate on behalf of the inputs. The grant arbiters get the request information from the input queues and make arbitration. Then, the arbitration result is sent to the accept arbiters. After

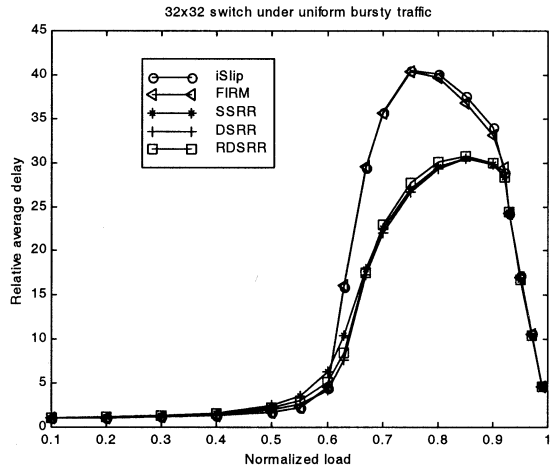

Fig. 4. Graph of average delay as a function of normalized load for a 32x32 switch under bursty traffic

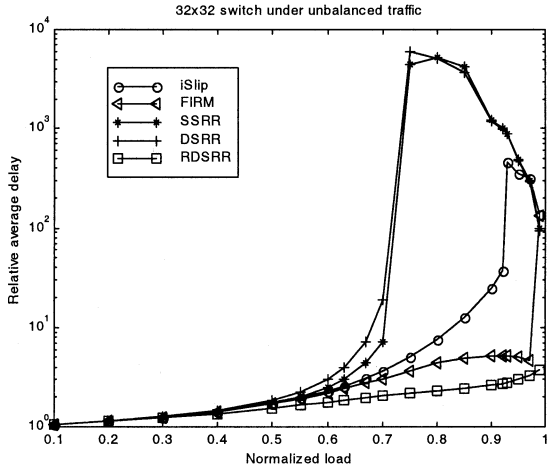

Fig. 6. Graph of average delay as a function of normalized load for a 32x32 switch unbalanced traffic

the input arbitration is made by the accept arbiters, the final result is put into a decision register. For pointer updating, it is even simpler than iSlip and FIRM, because the pointers are always incremented by 1, no matter what grant/accept is sent or whether the grant is accepted or not. So the updating of output pointers (which is done in the grant arbiters) will not depend on the accept signals generated by the accept arbiters. It makes the design of grant arbiters really simple.

To add the rotating enhancement function, we take the current time as a control of the rotating direction of both grant and request arbiters. It adds only a little complexity to the design.

VI. Conclusions

In this paper, a group of practical scheduling algorithms for VOQ switch architecture - Static Round-Robin (SRR) matching is introduced. It has three versions: single-SRR (SSRR), double-SRR (DSRR) and rotating DSRR (RDSRR). SSRR and DSRR achieve much better performance than the other algorithms, iSlip and FIRM, that we have considered under various traffic models, with even simpler hardware implementation. With an enhancement of "rotating pointers", RDSRR improves upon SSRR and DSRR. It solves the unfairness problem and guarantees better performance under any traffic pattern.

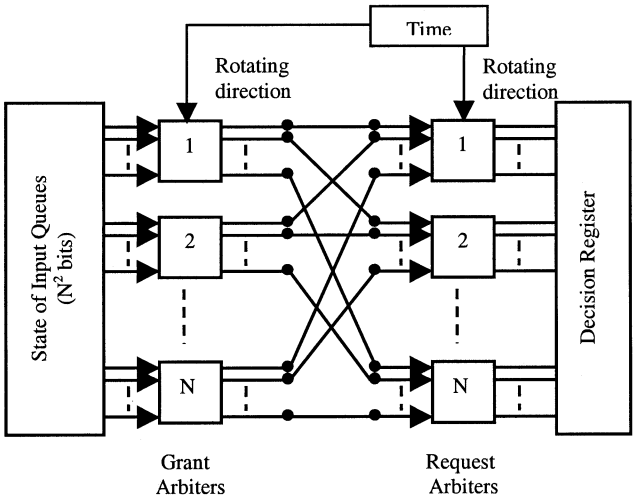

Fig. 7. Implementation of SRR Schemes

References

[1] M.J. Karol, M. G. Hluchyj, and S.P. Morgan, "Input Versus Output Queuing on a Space-Division Packet Switch," *IEEE Transactions on Communications*, 35:1347-56, 1987.

[2] T. Anderson, S. Owicki, J. Saxe, and C. Thacker, "High Speed Switch Scheduling for Local Area Networks," *ACM Trans. Comput. Syst.*, pp. 319-52, Nov. 1993.

[3] N. Mckeown, A. Mekkittikul, V. Anantharam, and J. Walrand, "Achieving 100% Throughput in an Input-Queued Switch," *IEEE Transactions on Communications*, 47: 1260-67, Aug. 1999.

[4] A. Mekkittikul, and N. Mckeown, "A Practical Scheduling Algorithm to Achieve 100% Throughput in Input-Queued Switches," *IEEE INFOCOM 98*, San Francisco, April 1998.

[5] N. McKeown, "Scheduling Cells in an Input-Queued Switch," PhD thesis, University of California at Berkeley, May 1995.

[6] H. J. Chao, and J.-S. Park, "Centralized Contention Resolution Schemes for A Large-Capacity Optical ATM Switch," *Proc. IEEE ATM Workshop*, Fairfax, VA, May 1998.

[7] D. N. Serpanos, and P. I. Antoniadis, "FIRM: A Class of Distributed Scheduling Algorithms for High-speed ATM Switches with Multiple Input Queues," *IEEE INFOCOM* 2000.

[8] R. E. Tarjan, "Data Structures and Network Algorithms," *Society for Industrial and Applied Mathematics*, Pennsylvania, Nov. 1983.

[9] J. E. Hopcroft, R. M. Karp, "An $n^{5/2}$ Algorithm for Maximum Matching in Bipartite Graphs," *Society for Industrial and Applied Mathematics Journal of Computation*, vol.2 pp.225-31, 1973.

[10] N. McKeown, M. Izzard, A. Mekkittikul, and M. Horowitz, "The Tiny Tera: A Small High-Bandwidth Packet Switch Core," *IEEE Micro Magazine*, vol.17, No. 1, pp. 26-33, Jan-Feb, 1997.