Stable and Practical Scheduling Algorithms for High Speed Virtual Output Queuing Switches

Jing Liu, Mounir Hamdi and Qingsheng Hu
Department of Computer Science
Hong Kong University of Science and Technology
Clear Water Bay, Kowloon, Hong Kong
liujing@cs.ust.hk, hamdi@cs.ust.hk and eehugs@ust.hk

Abstract

High-performance input queued switches achieve good performance with low cost. However, with the appearance of optical techniques, the line rate is much higher than before. Scheduling algorithms require not only good performance in delay and stability but fast speed and simple implementation as well. A variety of scheduling algorithms for Virtual Output Queuing (VOQ) packet switch architecture are proposed. Round-robin scheduling algorithms are fast and simple to implement in hardware. In particular, a group of fully desynchronized round-robin scheduling algorithms— SRR (static round robin matching), proposed recently, achieve pretty good delay performance while easy to implement. However, they are not stable under non-uniform traffic. Randomized algorithms are stable under any admissible traffic, however their delay is high and hardware implementation is complex. Based on the concept of randomized algorithms and SRR, we propose a group of new scheduling algorithms, DRDSRR, the improved version of DRDSRR, ARDSRR and the variations of ARDSRR. They not only ensure stability but also have good performance and simple implementation. We have proved stability in the paper.

1. Introduction

Nowadays, input queued (IQ) switches have become dominant in high speed switching. IQ switches only have an internal speedup equivalent to the line rate. However, it is well known that Head of Line Blocking (HOL) limits the throughput of an IQ switch with crossbar fabric to 58.6% [1] under uniform traffic when a single FIFO queue is deployed at each input. An alternative architecture, the virtual output queuing (VOQ) [2] can eliminate HOL blocking entirely while achieving scalability. Instead of a single FIFO queue, separate queues for different outputs are maintained at each input.

Variable scheduling algorithms are proposed. Maximum weight matching (MWM) algorithms perform well and can achieve 100% throughput, i.e. stable under any traffic, for example, LQF, OCF, LPF [3][4], etc. However they are impractical and too complex to implement in hardware. Maximal size matching (MSM) algorithms are practical and perform well under uniform traffic. However they are not stable under non-uniform traffic, such as iSLIP [5], FIRM [6], etc. Recently, a group of fully desynchronized round-robin scheduling algorithms, SRR (static round-robin) [7] are proposed. They perform much better than iSLIP and FIRM and are shown to achieve a low delay performance. However, they are inevitably unstable under non-uniform traffic, since they are MSM algorithms.

Randomized algorithms are stable under any traffic and only with linear complexity. However their delay is high compared with MSM algorithms. This is true even for non-uniform traffic as long as the maximal size matching algorithms are operating in their "stable" region [12]. The main reason for this is that the randomized algorithms have been designed with the objective of making them stable, rather than achieving a small average delay. Another problem with randomized algorithm is that they are still complex in hardware implementation. For example the "MERGE" procedure in LAURA and SERENA may induce large delay that degrade the performance of the algorithms [9]. In this paper, we have attempted to exploit the advantages of both randomized algorithms and SRR. We propose a group of algorithms, which are not only stable under any traffic but also have low delay and simple to implement as well.

The rest of paper is organized as follows: Section 2 introduces the round-robin scheduling algorithms — SRR. In Section 3, we discuss randomized algorithms. We present the DRDSRR algorithm and its improved version in Section 4. In Section 5, ARDSRR is proposed and also some variations of ARDSRR are provided. Finally, Section 6 concludes the paper.

2. A group of fully desynchronized round-robin scheduling algorithms

All existing round-robin scheduling algorithms are run with several iterations and each iteration consists of three steps: request, grant and accept.

Based on the observation of desynchronization of pointers leading to good performance, recently, a group of new round-robin scheduling algorithms— SRR (static round-robin) are proposed. The basic idea is to keep full pointer desychronization. There are different variations of SRR, which are SSRR (single static round-robin), DSRR (double static round-robin) and RDSRR (rotating DSRR) [7]. Among them, RDSRR performs the best. In RDSRR, the initialization of pointers are kept fully desynchronized. Both the input pointers and output pointers are set to some initial pattern such that there is no duplication among the pointers. At each time slot, pointers

Proceedings of the Eighth IEEE International Symposium on Computers and Communication (ISCC'03)
1530-1346/03 $17.00 © 2003 IEEE

of both input arbiters and output arbiters are always incremented by one (modulo N).

In the second step of Grant, to achieve fairness, clockwise and counter-clockwise rotations are done alternatively, each for one time slot, to search for the input to send grant.

The achievements with RDSRR are: (a) Lower delay (b) With clockwise and counter-clockwise rotation scheme, each input has a chance to be served. (c) Easy to implement in hardware.

3. Randomized algorithms

Randomized algorithms are based on several observations: (a) the state of the switch (for example, the lengths of its queues) changes little during successive time slots. This indicates it is possible to use the matching at time t for deriving the matching at time $t + 1$. (b) A randomly generated matching by a good scheme can be used to improve the matching used at time t for obtaining the matching at time $t + 1$.

3.1 Algo1: a randomized scheme with memory

Based on the above concepts, a basic randomized algorithm is proposed by Tassiulas [8].

Algo1:

(a) Let $S(t)$ be the schedule used at time t.

(b) At time $t + 1$ choose a matching $R(t + 1)$ uniformly at random from the set of all $N!$ possible matchings.

(c) Let $S(t + 1) = \arg \max_{S \in \{S(t), R(t+1)\}} <S, Q(t+1)>$ ($Q(t+1)$ is the queue-lengths matrix at time $t + 1$.)

Lemma 1 (Tassiulas [8]). Algo1 *is stable under any Bernoulli i.i.d. admissible input.*

3.2 Hamiltonian walk on the set of all matchings

We construct a graph with $N!$ nodes, each corresponding to a distinct matching, and all possible edges between these nodes. A Hamiltonian walk on this graph is to visit each of the $N!$ nodes exactly once during times $t = 1,\ldots, N!$. Extend $t > N!$ by defining $Z(t) = Z(t \bmod N!)$. For example, when $N = 3$, we obtain such a Hamiltonian walk: $Z(1) = (1, 2, 3)$, $Z(2) = (1, 3, 2)$, $Z(3) = (3, 1, 2)$, $Z(4) = (3, 2, 1)$, $Z(5) = (2, 3, 1)$, $Z(6) = (2, 1, 3)$, $Z(7) = Z(1)$, and $Z(8) = Z(2),\ldots$ ($(1, 2, 3)$ denotes a permutation $(\pi(1), \pi(2), \pi(3))$).

3.3 Algo2: a derandomized algorithm of Algo1

Based on the concept of Hamiltonian walk on the set of all matchings, Algo2 — a derandomization of Algo1 is proposed by Paolo Giaccone [9].

Algo2:

(a) Let $S(t)$ be the schedule used at time t.

(b) At time $t + 1$ let $R(t + 1) = Z(t + 1)$, the matching visited by the Hamiltonian walk.

(c) Let $S(t + 1) = \arg \max_{S \in \{S(t), R(t+1)\}} <S, Q(t+1)>$

Lemma 2 (Paolo Giaccone [9]) *Consider an input-queued switch with admissible Bernoulli i.i.d. inputs. Let Q(t) be the queue-size process that results when the switch uses scheduling algorithm B. Let $W^B(t)$ denote the weight of the schedule used by B at time t, and let $W^*(t)$ be the weight of MWM given the same queue-size process Q(t). If there exists a positive constant c such that the property*

$W^B(t) \geq W^*(t)$*-c holds for all t, then the algorithm B is stable.*

Lemma 3 (Paolo Giaccone [9]) *An input-queued switch using* Algo2 *is stable under all admissible Bernoulli i.i.d. inputs.*

Compare Algo2 with Algo1, derandomization is used in Algo2, which is much easier to implement in hardware than random scheme.

4. DRDSRR algorithm

In Section 3, there is a proof for the stability of Algo2. We can see that when an algorithm uses memory and the Hamiltonian walk — the derandomzation scheme, it is stable. This property will be used as part of the design of our DRDSRR (derandomized RDSRR) algorithm.

4.1 Specification of DRDSRR

By the above observations, we now give the specification of DRDSRR:

Initialization. The output pointers are set to some initial pattern such that there is no duplication among the pointers. The same is done for the input pointers.

Step 1: Let $S(t - 1)$ be the schedule used at previous time slot. At current time slot t, let $R(t) = Z(t)$, the matching visited by the Hamiltonian walk.

Step 2: Request. Each input sends a request to every output for which it has a queued cell.

Step 3: 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. To achieve fairness, clockwise and counter-clockwise rotations of the pointers are done alternatively, each for one time slot. 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 4: Temporal Accept. If an input receives a grant, it selects 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 a grant or not.

Step 2 to *Step 4* are iteratively executed, resulting in a matching, we call it $M'(t)$ (unmatched inputs are matched to unmatched outputs arbitrarily with no weight).

Step 5: Accept. Let $M(t) = \arg \max_{S \in \{S(t-1), R(t), M'(t)\}} <S, Q(t)>$.

$M(t)$ is the present schedule. The corresponding inputs send accepts to corresponding outputs.

4.2 Stability of DRDSRR

Proof. DRDSRR uses the Hamiltonian walk with memory, $<M(t),Q(t)> \geq <M(t-1),Q(t)>$ and $<M(t),Q(t)> \geq <Z(t),Q(t)>$. Therefore, **Lemma 2** and **Lemma 3** apply, this is sufficient to prove its stability.

4.3 An improved version of DRDSRR

The RDSRR scheduling algorithm with multiple iterations can result in a maximal matching. However, the matchings determined by DRDSRR and its variations are not of maximal size. Queue-lengths are only used to select the heaviest matching. It is therefore possible that the resulting matching is heavy, but not of maximal size. Therefore, we can make DRDSRR a maximal size matching. Suppose there are k unmatched inputs and outputs left by the algorithm. We augment the matchings between those inputs and outputs repeatedly until no more connections can be made. This is easy to implement. The time complexity is at most $O(k^2)$. The maximal version of DRDSRR is named as DRDSRR(v2), Its performance is comparable with MWM.

The complexity of DRDSRR is lower than other approximate MWM algorithms [13] and possible hardware design is given in [13].

4.4 Simulation results

In our simulation, we consider a 32x32 switch. The traffic is Bernoulli i.i.d. and admissible. Uniform traffic, uniform bursty traffic and various non-uniform traffic patterns, namely the diagonal and hotspot cases are considered.

The traffic matrix of hotspot traffic is like (4x4 switch):

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

Output 1 is the hot-spot with higher rate of traffic destined to it, and all other traffic is distributed to other outputs uniformly.

The traffic matrix of the diagonal traffic is like (4x4 switch):

$$\begin{bmatrix} x & 1-x & 0 & 0 \\ 0 & x & 1-x & 0 \\ 0 & 0 & x & 1-x \\ 1-x & 0 & 0 & x \end{bmatrix}$$

The traffic is concentrated on two diagonals. One is heavier than the other. (x = 2/3)

DRDSRR and DRDSRR(v2) have good performance under both uniform traffic and uniform bursty traffic [13]. They have the advantage of round-robin scheduling algorithms. For non-uniform traffic, they have much better performance than other algorithms.

Figure 1 shows the results under diagonal traffic. DRDSRR and DRDSRR(v2) have much better performance than all the other algorithms. Surprisingly, their performance is comparable with MWM.

Figure 2 shows the results under hotspot traffic. DRDSRR has good performance as well. Especially under high load, DRDSRR has a much lower delay than all the other algorithms. DRDSRR(v2) has even better performance than DRDSRR and it is comparable with MWM.

5. ARDSRR algorithm

In Section 4, we propose DRDSRR and its improved version, DRDSRR(v2) to achieve low delay. Howeimplementation even simpler while still maintain good performance?

We have several observations.

First, in DRDSRR, the last step compares the weight of three matchings, namly the matching of last time slot, the matching from Hamiltonian walk and the matching from current RDSRR

Figure 3 shows the contributions of these three matchings,

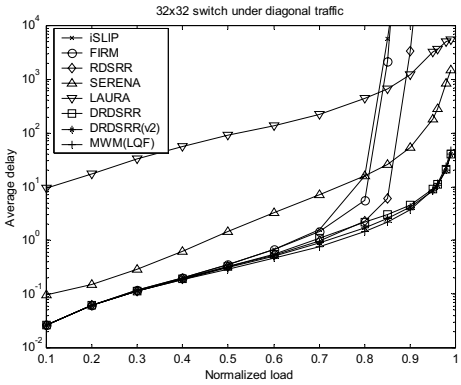

Figure 1. Average delay under diagonal traffic.

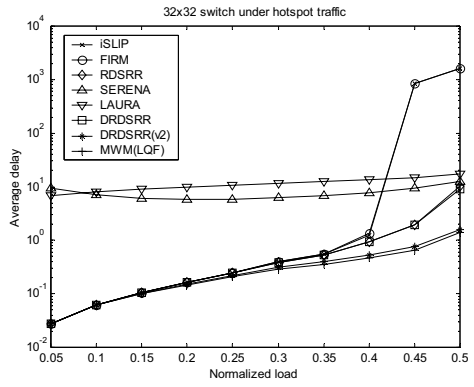

Figure 2. Average delay under hotspot traffic.

i.e. in which percentage, each matching is selected as the final scheduling matching.

When the load is light, almost all the matchings selected are from RDSRR scheduling. When the load is heavy, more and more matchings are derived from the matching of last time slot, since heavy load results in the accumulation of queue lengths and the heavy queues with large possibility remain heavy in successive time slots. In all ranges, very little percentage is derived from Hamiltonian walk matching.

Second, the accumulation of queue lengths is due to arrivals and the arrival process is also a source of randomness.

Based on the above observations, we devise a new algorithm, termed as ARDSRR (arrival RDSRR). In this Algorithm, we remove the use of Hamiltonian walk matching in order to make the hardware implementation even simpler and exploit the use of arrival information in obtaining RDSRR matching to achieve good performance.

5.1 The specification of ARDSRR

Initialization. The output pointers are set to some initial pattern such that there is no duplication among the pointers. The same is done for the input pointers.

Step 1: Let $S(t - 1)$ be the schedule used at previous time slot.

Step 2: Request. Each input sends a request to every output for which it has a queued cell.

Step 3: Grant. If an output receives any requests,

Case 1: If there are requests with a new arrival, it chooses the one with a new arrival that appears next in a fixed, round-robin schedule starting from the highest priority element.

Case 2: If there is no request with a new arrival, it chooses the one as usual, i.e. the one that appears next in a fixed, round-robin schedule starting from the highest priority element.

To achieve fairness, clockwise and counter-clockwise rotations of the pointers are done alternatively, each for one time slot. 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 4: Temporal Accept. If an input receives a grant,

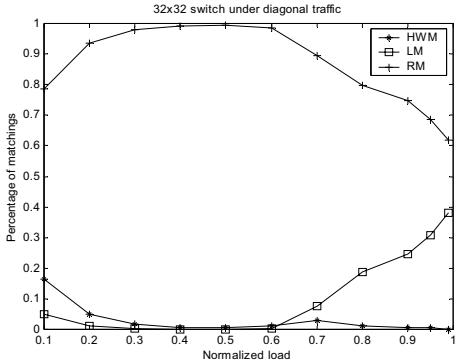

Figure 3. Percentage of different matchings selected as final scheduling under diagonal traffic.

Case 1: If there is a grant with a new arrival (at most one), it selects it as temporal accept.

Case 2: If there is no grant with a new arrival, it selects 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 a grant or not.

Step 2 to *Step 4* are iteratively executed, resulting in a matching, we call it $M'(t)$ (unmatched inputs are matched to unmatched outputs arbitrarily with no weights).

Step 5: Accept. Let $M(t) = \arg \max_{S \in \{S(t-1), M'(t)\}} <S, Q(t)>$. $M(t)$ is the present schedule. The corresponding inputs send accepts to corresponding outputs.

5.2 Stability of ARDSRR

Proof. Follow from **Lemma 1**, and observe the randomness in choosing the matching is in the grant and temporal accept steps. The arrival is also a source of randomness. This is sufficient to ensure its stability.

5.3 A variation of ARDSRR: PARDSRR

We find some observations in ARDSRR.

1) If input i_1 has a new arrival to output j_1, and at the same time input i_1 has a request with no arrival to output j_2. If output j_1 grants input i_1, then the request from i_1 to j_2 will certainly not be accepted. We should avoid letting output j_2 grant input i_1. In stead, if output j_2 grants other inputs, it will increase the instant throughput.

2) For an output j, if there are two requests from input i_1 and input i_2, but input i_1 has a new arrival with the request and input i_2 has no arrival with the request. For sure the request from input i_2 will not be granted. We can avoid sending request from input i_2 to output j.

We can improve ARDSRR by taking request with a new arrival as higher priority over those without arrivals and deal with them first. As a result, we name the new scheme as PARDSRR (priority arrival RDSRR). The specification of PARDSRR is as following:

Initialization. The output pointers are set to some initial pattern such that there is no duplication among the pointers. The same is done for the input pointers.

Step 1: Let $S(t - 1)$ be the schedule used at previous time slot.

Step 2: Request. Each input sends a request to every output for which it has a queued cell. If there is a new arrival, the input issues an arrival signal to the output.

Step 3: Matching with arrivals. Check each output. If there are requests with arrivals, the output chooses the one with a new arrival that appears next in a fixed, round robin schedule starting from the highest priority element and sends grant to the

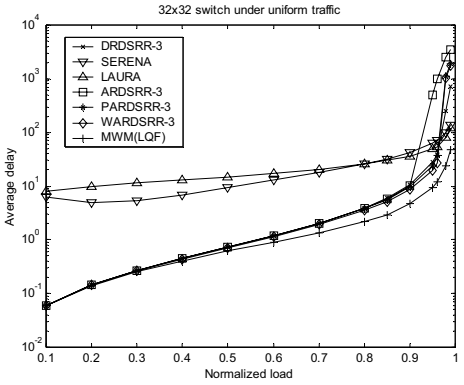

Figure 4. Average delay under uniform traffic.

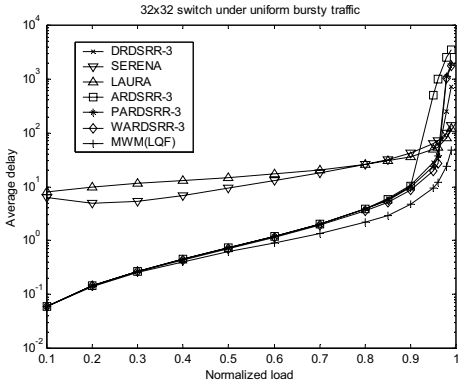

Figure 5. Average delay under uniform bursty traffic.

corresponding input. The input which receives grant, sends a temporal accept.

Step 4: Matching without arrivals. Consider all the unmatched inputs and outputs left by *Step 3*. Do as usual RDSRR scheduling. We obtain a matching from *Step 3* and this step, we call it $M'(t)$ (unmatched inputs are matched to unmatched outputs arbitrarily with no weights).

Step 5: Accept. Let $M(t) = \arg \max_{S \in \{S(t-1), M'(t)\}} <S, Q(t)>$. $M(t)$ is the present schedule. The corresponding inputs send accepts to corresponding outputs.

5.4 A variation of ARDSRR: WARDSRR

In WARDSRR (weighted arrival RDSRR), we make a modification in *Step 3* of PARDSRR. In PARDSRR, a round-robin scheme is used when selecting matches with arrivals. WARDSRR will select the matches with arrivals by considering the queue length. *Step 3* of WARDSRR is as following:

Step 3: Matching with arrivals. Check each output. If there are requests with arrivals, the output chooses the one with a new arrival that has the heaviest weight and sends grant to the corresponding input. The input, which receives grant, sends a temporal accept.

5.5 Simulation results

Figure 4 shows the results under uniform traffic. All the round-robin scheduling algorithms are run with 3 iterations. ARDSRR is not good under high load. However, PARDSRR and WARDSRR are much better than ARDSRR, since the instant throughput increases. They are comparable with DRDSRR.

Figure 5 shows the results under uniform bursty traffic. Again, all the round-robin scheduling algorithms are run with 3 iterations. The performance of ARDSRR, PARDSRR and WARDSRR are similar to the case of uniform traffic.

Figure 6 compares all the algorithms under diagonal traffic. ARDSRR, PARDSRR and WARDSRR are not as good as DRDSRR. However, PARDSRR and WARDSRR are better than ARDSRR. They have much lower delay than SERENA and LAURA when the load is under 0.9. In the range of higher load, they are very close to SERENA and still better than LAURA.

Figure 7 shows the results under hotspot traffic. ARDSRR is not very good under high load. However, PARDSRR and WARDSRR perform very well. They have much lower delay than SERENA and LAURA, even much better than DRDSRR. Surprisingly, their performance is comparable with MWM. Also, WARDSRR is slightly better than PARDSRR.

5.6 Complexity analysis and hardware implementation

PARDSRR and WARDSRR perform well and have simpler implementation than DRDSRR, since they avoid the implementation of Hamiltonian walk matching. Their difference in computing matching $M'(t)$ will not increase the complexity of implementation.

In PARDSRR, we only need to issue arrival signals along with sending requests and in the grant step, let the first iteration deal with requests with arrivals in round-robin, and the following iterations do as usual.

In WARDSRR, since there is at most one arrival from each input at one time slot, all the outputs together receive at most N arrival signals, which means at most the comparisons cost

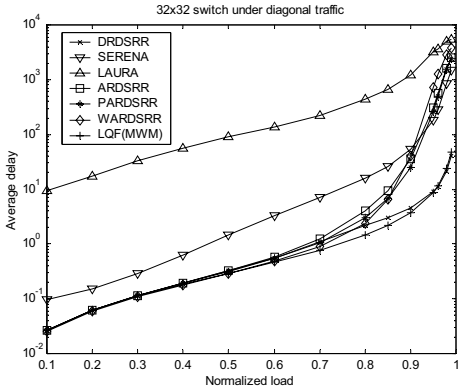

Figure 6. Average delay under diagonal traffic.

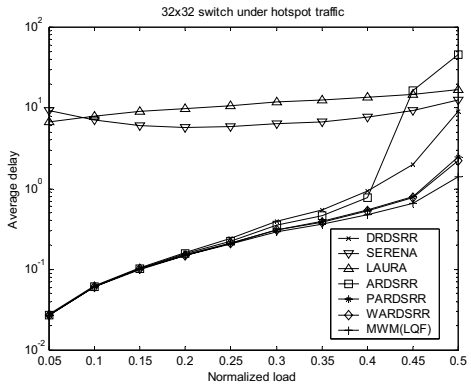

Figure 7. Average delay under hotspot traffic.

$O(\log N)$ time. The worst case is that all the inputs have arrivals destined to the same output. Then this output needs $O(\log N)$ time to obtain the one with the heaviest weight. On average, each output requires less time and the comparison can be done in parallel.

Figure 8 shows the hardware implementation of PARDSRR. Compared with the implementation of DRDSRR, The module for implementing Hamiltonian walk matching is avoided and RDSRR scheduler is modified a little bit.

Figure 9 shows how to implement the modified RDSRR scheduler. $2N$ arbiters for each input/output and $2N^2$-bit memory are used. Among the memory, N^2-bit is used for registering requests and another N^2-bit is used for registering arrivals.

First, deal with arrival signals. When an input or output is matched, its arbiter is disabled. This is simple for both the grant and accept arbiter. Whenever the grant arbiter makes a decision, it is disabled since the corresponding input has at most one arrival, the grant will certainly be accepted. Whenever the accept arbiter receives a grant with arrival, it is disabled. The following steps do as usual RDSRR scheduling.

6. Conclusion

Maximum weight matching algorithms perform very well under non-uniform traffic, and are consequently stable. However they are too complex to implement. Randomized algorithms are shown to achieve stability under any admissible traffic, but they induce a high delay compared with iterative maximal matching algorithms and still complex in hardware implementation. The group of SRR maximal matching algorithms have good delay performance, but are not stable under non-uniformtraffic. In this paper, we have proposed a group of algorithms, i.e. DRDSRR, its improved version, ARDSRR and its variations, based on the concept of randomized algorithms and RDSRR. They are all shown to be stable under any admissible traffic while maintaining lower delay performance, especially under non-uniform traffic. They are also simple to implement, which makes them practical. We have also demonstrated possible hardware design of these algorithms.

7. References

[1] M. Karol, M. Hluchyj, and S. Morgan, "Input versus Output Queuing on a Space Division Switch," *IEEE Trans. Communications*, 35(12) (1987) pp.1347-1356.

[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] A. Mekkittikul and N. McKeown, "A Starvation-free Algorithm for Achieving 100% Throughput in an Input-Queued Switch," *ICCCN '96,* Oct. 1996, pp.226-231.

[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, vol. 2, pp.792-799.

[5] N. McKeown, "iSLIP: A Scheduling Algorithm for Input-Queued Switches," *IEEE Transactions on Networking,* April 1999, Vol 7, No.2.

[6] 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.

[7] Ying Jiang and M. Hamdi, "A Fully Desynchronized Round-Robin Matching Scheduler for a VOQ Packet Switch Architecture," *High Performance Switching and Routing*, 2001 *IEEE Workshop on*, pp. 407-411.

[8] L. Tassiulas, "Linear Complexity Algorithms for Maximum Throughput in Radio Networks and Input Queued Switches," *IEEE INFOCOM'98*, New York, 1998, vol. 2, pp.533-539.

[9] P. Giaccone, B. Prabhakar, and D. Shah, "Towards Simple, High-performance Schedulers for High-aggregate Bandwidth Switches," *IEEE INFOCOM*, 2002.

[10] A. Nijenhuis and H. Wilf, "Combinatorial Algorithms: for Computers and Calculators", 2nd *Edition*, *Academic Press*, chap. 7, New York, 1978, p. 56.

[11] A. Mekkittikul, "Scheduling Non-uniform Traffic in High Speed Packet Switches and Routers," *PhD Thesis (181 pages),* Stanford University, November 1998.

[12] P. Giaccone, B. *Queueing and Scheduling Algorithms for High-Performance Routers*. PhD thesis, Politecnico Di Torino, Italy, 2002.

[13] J. Liu, H. C. Kit, M. Hamdi and C. Y. Tsui, "Stable Round-Robin scheduling Algorithms for High-Performance Input Queued Switches", *HOT Interconnects 10*, 2002.

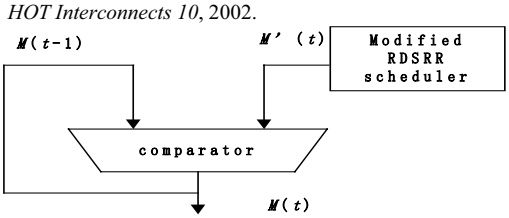

Figure 8. Implementation of the PARDSRR scheme.

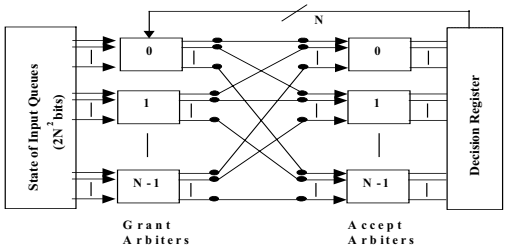

Figure 9. Implementation of the modified RDSRR scheduler.