Stable Round-Robin Scheduling Algorithms for High-Performance Input Queued Switches[1]

Jing Liu, Hung Chun Kit¶, Mounir Hamdi, and Chi Ying Tsui¶
Department of Computer Science
¶Department of Electrical and Electronic Engineering
Hong Kong University of Science and Technology
Clear Water Bay, Kowloon, Hong Kong
hamdi@cs.ust.hk

Abstract

High-performance input-queued switches require high-speed scheduling algorithms while maintaining good performance. Various round-robin scheduling algorithms for Virtual Output Queuing (VOQ) crossbar-based packet switch architectures have been proposed. It has been demonstrated that they can operate at high speed (e.g., OC192), and are relatively simple to implement in hardware. In particular, a group of fully desynchronized round-robin scheduling algorithms, named SRR (static round robin matching), which have been proposed recently, achieve pretty good delay performance while easy to implement. The main problem with these arbitration algorithms is that they are not stable under non-uniform traffic. In this paper, based on the concept of both randomized algorithms and SRR, we propose a new scheduling algorithm, termed DRDSRR (derandomized rotating double static round-robin), which is shown to be stable under all Bernoulli i.i.d. admissible traffic and performs better than SRR.. In addition, we also propose a novel pipelining scheme for the hardware implementation of these scheduling algorithms which can achieve one more iteration within each cycle time, and hence better performance, when compared with the pipelining schemes used in conventional designs.

1. Introduction

Input queued (IQ) switches are frequently used in the design of high-speed switching architectures. IQ switches have an internal speedup equivalent to the line rate, and as a result, require moderate memory bandwidth which is commercially available. However, it is well known that the Head of Line Blocking (HOL) limits the throughput of an IQ switches with a crossbar fabric to a 58.6% [1] under uniform traffic when a single FIFO queue is used at each input. It is well established that using virtual output queuing (VOQ) [2] can eliminate the HOL blocking entirely while retaining the same scalability and memory bandwidth requirement of a FIFO IQ switch. Instead of a single FIFO queue, separate queues for different outputs are maintained at each input. With a proper scheduling algorithm, it has been shown that 100% throughput can be achieved for an IQ switch employing VOQ under both uniform and non-uniform traffic.

Various scheduling algorithms have been proposed for the VOQ architecture. For example, maximum weight/size matching algorithms can achieve 100% throughput, i.e., stable under any admissible traffic. Such algorithms include LQF, OCF, LPF [3][4]. But they are impractical and too complex to implement in hardware for high-speed line rates. Maximal size matching (MSM) algorithms are practical and perform well under uniform traffic. But they are not stable under non-uniform traffic, which infers less than 100% throughput. Example algorithms are iSLIP [5], and FIRM [6]. Recently, a set of fully desynchronized round robin scheduling algorithms, SRR (static round-robin) [7] have been proposed. They perform much better than iSLIP and FIRM using even less complicated hardware. However, they are inevitably unstable under non-uniform traffic, since they are MSM algorithms.

Recently, it has been proven that some randomized algorithms can be made stable under any admissible traffic and only require linear complexity [8]. However, through extensive experimentations, it has been shown that the average delay of these randomised algorithms is higher than those of maximal size matching 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 randomised algorithms have been designed with the

[1] This research work has been partially supported by grant from the Hong Kong Research Grant Council.

objective of making them stable, rather than achieving a small average delay. In this paper, we propose a new algorithm: DRDSRR (derandomized rotating double static round-robin) that attempts to combine the advantages of both schemes. That is, being stable and having lower average delays. DRDSRR achieves both a high instantaneous throughput of a MSM algorithm, and the stability of randomized algorithms. As a result, it will be shown that the DRDSRR is not only stable under any traffic, but has low average delay as well.

The rest of the paper is organized as follows. Section 2 briefly describes randomized algorithms. Section 3 introduces the SRR scheduling algorithms. We present the DRDSRR algorithm in Section 4. In Section 5, an improved version of DRDSRR is proposed. Section 6 presents the simulation results and Section 7 analyzes the computational complexity of the proposed algorithms. Section 8 gives a possible hardware implementation of our algorithms. In Section 9, we exploit a generalization of DRDSRR to other round-robin scheduling algorithms. Section 10 concludes the paper.

2. Randomized algorithms

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

2.1. Algo1: A randomized scheme with memory

Based on the above concepts, a scheduling (matching) 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.*

2.2. 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 get 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 $(\pi(1), \pi(2), \pi(3))$).

2.3. Algo2: A derandomization of Algo1

Based on the concept of Hamiltonian walk on the set of all matchings, Algo2 — a derandomization of Algo 1 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 a given 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 Maximum Weight Matching (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.*

Proof. Since there is at most 1 packet arriving at or departing from each input queue in each time slot, we obtain for any matching M

$$<M,Q(t)> \geq <M,Q(t - s)> - sN. \quad (1)$$

Let $S(t)$ denote the schedule used by Algo2 at time t, and let $W^2(t) = <S(t),Q(t)>$ be its weight.

Consider a specific time instant T. Let S_1 and S_0 denote the maximum weight matchings at time T and $T - N!$, respectively. By the property of the Hamiltonian walk, there is a $t' \in [T - N!, T]$ such that $Z(t') = S_0$. Then

$<S(t'),Q(t')> \geq <S_0,Q(t')>$ (the definition of Algo2)

$<S_0,Q(t')> \geq <S_0,Q(T - N!)>-(t' + N! - T)N$ (from (1))

It follows from (1) and the definition of Algo2 that

$<S(t),Q(t)> - N \leq <S(t),Q(t + 1)> \leq <S(t + 1),Q(t + 1)>$.

Using this repeatedly in the following, we obtain

$<S(T),Q(T)> \geq <S(t'),Q(t')> - (T - t')N \geq <S_0,Q(T - N!)> - NN! \geq <S_1,Q(T - N!)> - NN!$

(S_0 is the maximum weight schedule at time (T-N!)) and $<S_1,Q(T\text{-}N!)>\text{-}NN! \geq <S_1,Q(T)> - 2NN!$ (from (1))

Since T was arbitrary, we have shown that

$W^2(t) \geq W^*(t)\text{-}2NN!$ For every t.

By Lemma 2, it is proven that Algo2 is stable under all admissible Bernoulli i.i.d. inputs.

3. A group of Fully Desychchronized Round-robin Algorithms

Recently, a group of new round-robin scheduling algorithms SRR (static round-robin) has been proposed [7]. The basic idea is to keep full pointer-desychronization in the output and/or input arbiters. 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, the RDSRR performs the best. We now give the specification of RDSRR:

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

The 3 steps of a single 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. To achieve fairness, clockwise and counter-clockwise rotation of the arbiter 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 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 a grant or not.

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

4. Derandomized rotating double static round-robin algorithm

In this paper, we propose a new scheduling algorithm which not only maintains the good delay performance of RDSRR but is stable under any admissible traffic as well. We call it the DRDSRR (derandomized rotating double static round-robin).

In Section 2, we briefly introduced a proof for the stability of Algo2. In particular, we can see that if an algorithm uses memory and the Hamiltonian walk, it can be made stable. This property will be used as part of the design of our DRDSRR algorithm.

4.1. Specification of DRDSRR

Through 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 the previous time slot. At the 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 weights).

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, $<M(t),Q(t) \geq <M(t\text{-}1),Q(t)>$ and $<M(t),Q(t)> \geq <Z(t),Q(t)>$. Therefore, Lemma 2 and Lemma 3 apply. The weight difference between the maximum weight matching and

DRDSRR is lower bounded by a positive constant all the time. As shown in Lemma 3, this is sufficient to prove its stability.

The Hamiltonian walk is used to ensure the stability of DRDSRR. In practice, the implementation of the Hamiltonian walk is extremely simple. In Chapter 7 of [10], there is a very simple algorithm generating a Hamiltonian walk and it only requires $O(1)$ space and $O(1)$ time.

5. An improved version of DRDSRR

The RDSRR scheduling algorithm with multiple iterations can result in a maximal matching. However, the matching determined by DRDSRR is not of maximal size. In *Step 5*, queue-lengths are only used to select the heaviest matching from $S(t-1)$, $R(t)$ and $M'(t)$. It is therefore possible that the resulting matching is heavy, but not of maximal size. We consider making DRDSRR a maximal size matching as well. Suppose there are k unmatched inputs and outputs left by the DRDSRR 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 $O(k^2)$. We shall call this version as DRDSRR(v2).

6. Simulation results

In our simulation, we consider a 32x32 switch. The traffic is Bernoulli i.i.d. and is admissible (no input or output is overloaded). Uniform traffic, uniform bursty traffic and various non-uniform traffic patterns, namely the diagonal and hotspot cases are considered. The algorithms are executed using 1 iteration.

The traffic matrix of hotspot traffic is like (for a 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 (for a 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)

6.1. Delay performance

Figure 1 shows the simulation results under uniform traffic. The average delay of DRDSRR is lower than FIRM and iSLIP and comparable with RDSRR. When the load is below 0.5, SERENA and LAURA have much higher delays than DRDSRR. When the load is above 0.5, SERENA and LAURA are better than DRDSRR and other maximal size matching algorithms. It is because those maximal size matching algorithms are run using only 1 iteration. However, DRDSRR(v2) has a much lower delay over all the ranges.

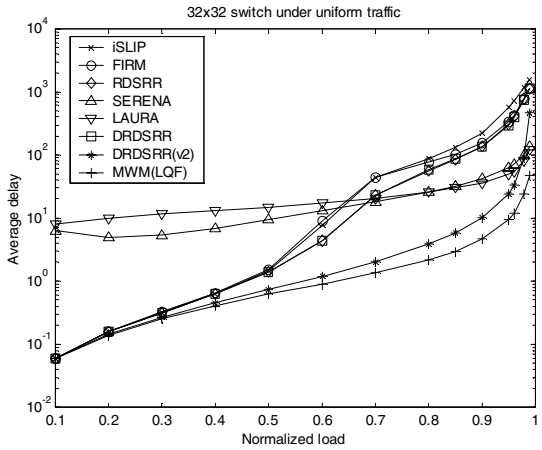

Figure 1. Average delay under uniform traffic.

Figure 2 shows the results when those maximal size matching algorithms, DRDSRR and DRDSRR(v2) are run using 3 iterations under uniform traffic. It is obvious that iSLIP, FIRM, RDSRR, DRDSRR and DRDSRR(v2) have similar delay. They are much better than SERENA and LAURA.

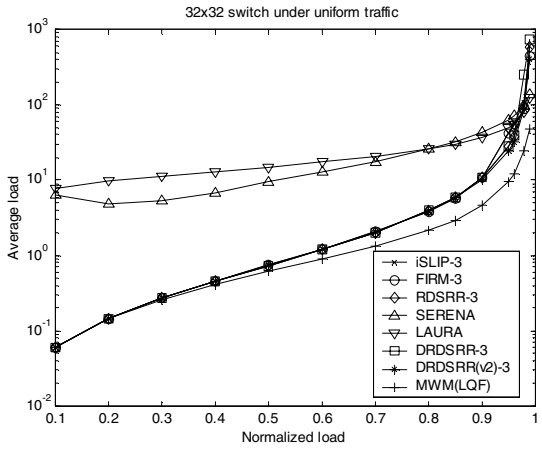

Figure 2. Average delay under uniform traffic.

Figure 3 compares the various algorithms under uniform bursty traffic. These results are similar to the uniform case and DRDSRR(v2) still performs very well.

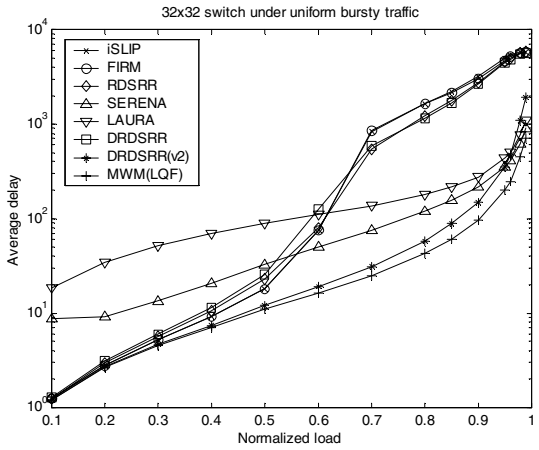

Figure 3. Average delay under uniform bursty traffic.

Figure 4 shows the results when all the maximal size matching algorithms, DRDSRR and DRDSRR(v2) are run using 3 iterations under uniform bursty traffic. We can see that the results are similar to the uniform case with 3 iterations.

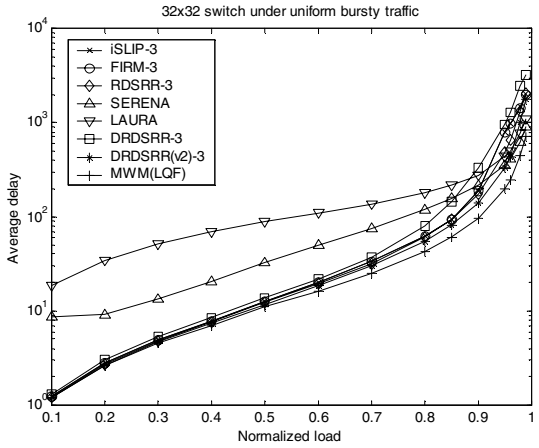

Figure 4. Average delay under uniform bursty traffic.

Figure 5 shows the results under diagonal traffic. DRDSRR and DRDSRR(v2) both have much better performance than FIRM, iSLIP, and RDSRR, even much better than SERENA and LAURA. Their performances are comparable with MWM.

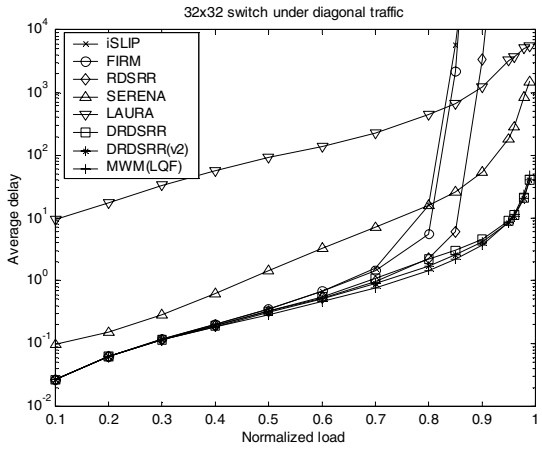

Figure 5. Average delay under diagonal traffic.

Figure 6 shows the results under hotspot traffic. DRDSRR has a good performance. Especially when 0.3 < load < 0.5, DRDSRR has a much lower delay than iSLIP and FIRM. Under the load of all ranges, DRDSRR is better than SERENA and LAURA. DRDSRR(v2) is even better than DRDSRR.

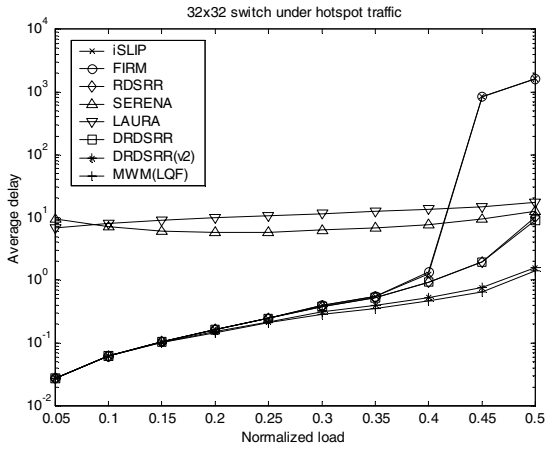

Figure 6. Average delay under hotspot traffic.

6.2. Stability under non-uniform traffic

iSLIP, FIRM and RDSRR are all stable under uniform traffic, but not under non-uniform traffic. DRDSRR and DRDSRR(v2) are stable under any admissible traffic. We now compare DRDSRR and DRDSRR(v2) with the others under diagonal traffic as a function of their norms of queue-lengths vector. We define queue-lengths vector as $Q(t) = (Q_{1,1}(t),\ldots,Q_{1,N}(t),\ldots,Q_{M,N}(t))^T$, the norm of queue-lengths vector as $\|Q(t)\| = \sqrt{Q^T(t)Q(t)}$.

From **Figure 7**, we see that iSLIP, FIRM and RDSRR will have large packet losses at high loads. DRDSRR and DRDSRR(v2) perform quite competitively with respect to

MWM. They maintain even much smaller queue lengths than SERENA and LAURA do.

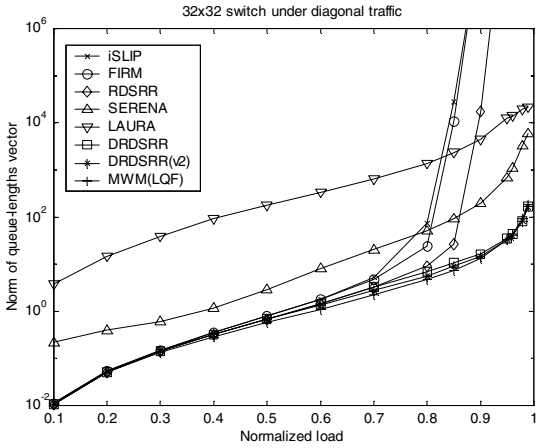

Figure 7. Norm of queue-lengths vector under diagonal traffic.

7. Complexity analysis

Actually, to make an iterative algorithm stable, there are other ways proposed. For example, iLQF [11], and iLPF [11]. Both algorithms attempt to approximate the MWM algorithms. iLQF is a 3-step iterative algorithm. In the Grant and Accept steps, each arbiter chooses the one with the largest queue length. Those arbiters dominate the iteration time, each requiring a modified N-input magnitude comparator, which are slow due to the large number of input values they need to compare. The basic building block, a two-input integer comparator is relatively complex. The best running time of each arbiter is $O(\log b \cdot \log N)$, where b is the number of bits of each input equalling to $\log Q_{\max}$. iLPF is simpler than iLQF. The double for-loop version first requires the use of a sorter to sort both the input port occupancies and output port occupancies. It takes an $O(N\log N)$ time and more space cost. The sorting procedure is followed by a double for-loop which chooses a maximal size matching. For our DRDSRR, except for the cost of obtaining the matching $M'(t)$ derived from the RDSRR, it only requires an additional computing of the Hamiltonian walk matching, which is simple to implement. The algorithm in [10] requires only $O(1)$ time and $O(1)$ space. Then we require to compute weights of the matchings, which cost $O(\log N)$ time using $\log N$ adders and get a matching with maximum weight among them, it only takes constant time. All together this is not high and space cost is low too. Concurrency between obtaining $R(t)$, $M'(t)$ and computing their weights are possible, allowing the complexity to be even lower.

DRDSRR(v2) requires to compute a maximal size matching. An additional computational complexity of $O(k^2)$ is required if the number of unmatched inputs and outputs is k. That is not a large additional expense.

8. Hardware implementation

Figure 8 shows a scheme for a possible design of the DRDSRR. The module for the RDSRR scheduler can be made similar to that used in the iSLIP algorithm (see Fig. 7 in [7]).

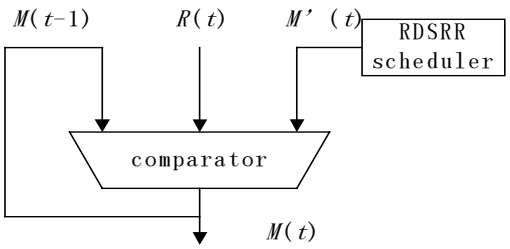

Figure 8. Implementation of the DRDSRR scheme.

Below, we will illustrate an improved scheme for the implementation of the RDSRR scheduler.

The advantage of the iterative maximal matching scheduling algorithms is that they are relatively easy to implement in hardware. **Figure 9** shows the overall architecture of the RDSRR scheduler. The three blocks represent the request phase, the grant phase, and the accept phase of the algorithm. The request blocks are used to store and forward the incoming request vectors to the grant blocks. After the request vectors pass through the grant stage and the accept stage, the scheduler will make a decision by selecting which input and output ports should be connected. The successive iterations of the algorithm can help increase the number of input/output matching during each cell time. A feedback loop is used to mask off those requests that have been accepted in the previous iterations. The scheduler will not consider the accepted requests again during the next iteration. In fact, the request blocks are registers, and we can simply treat them and the grant blocks as a single unit.

The throughput of the hardware architecture shown in **Figure 9** can be improved by applying efficient pipelining techniques. By careful examination on the pipeline structure used in the Tiny Tera (**Figure 10**) [13], we realized that the pipeline structure can be further optimized so that it can achieve one more iteration within the fixed amount of cycle time by using the same grant and accept arbiter hardware structure.

The optimized pipelining scheme can be applied in any iterative maximal matching scheduling algorithm that is made up of request, grant and accept phases.

In the following, we try to evaluate a generic maximal matching scheduling algorithm with the *original pipeline structure* which used in the Tiny Tera and the proposed *optimized pipelining scheme*. These two different pipelining schemes are being evaluated under Bernoulli i.i.d. uniform traffic, uniform bursty traffic and non-uniform traffic with a 128x128 switch executing one iteration only.

The traffic matrix of the non-uniform traffic is like (for a 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. (x = 1/2)

Figure 11 and **Figure 12** shows that the proposed optimized pipeline scheme gives a significant improvement under Bernoulli i.i.d. uniform and uniform bursty traffic when compared with the conventional pipeline scheme. While **Figure 13** shows the proposed optimized pipeline scheme gives a little improvement under the non-uniform traffic that is described above.

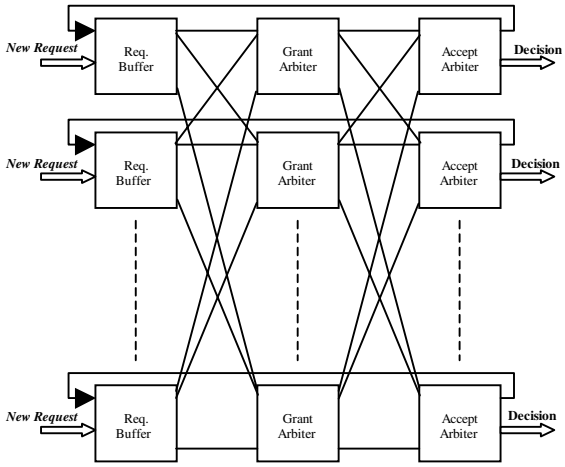

Figure 9. Overall architecture on the scheduler.

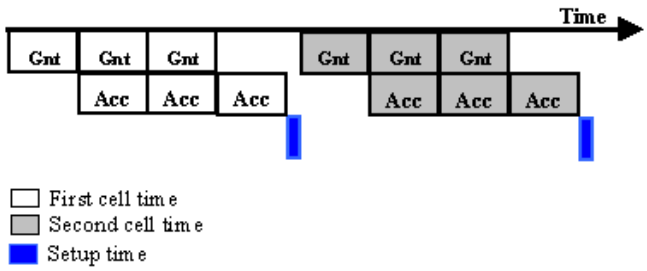

Figure 10. Pipeline Structure used in Tiny Tera.

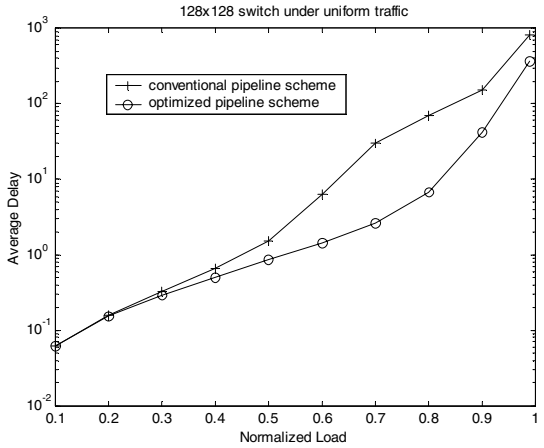

Figure 11. Average delay under uniform traffic.

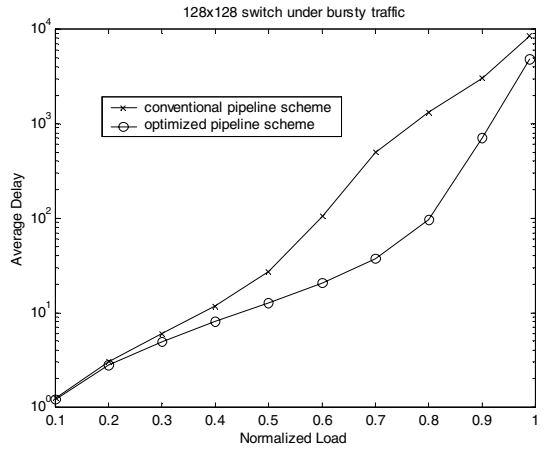

Figure 12. Average delay under bursty traffic.

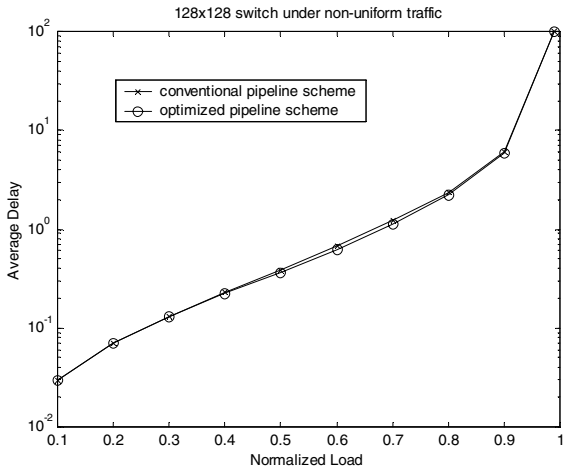

Figure 13. Average delay under non-uniform traffic.

The most critical components inside the scheduler are the grant arbiters and the accept arbiters. These two arbiters have the same hardware architecture as shown in **Figure 14** [13].

The priority filter and the priority encoder are sitting at the most critical path inside the scheduler. Therefore careful design on both of the priority filter and the priority encoder can save the data processing time, and hence detrmine the hardware scalability of the scheduler.

The priority filter is being built with time delay proportional to $\log_2 N$ (where N is the number of input port). All the hardware logic inside the priority filter are highly parallelized and can be easily built by using the AND, OR, XNOR and MUX logic.

The time delay for the priority encoder circuit is also proportional to $\log_2 N$ (where N is the number of input port). With the use of dynamic logic and parallelizing the logic, we have already demonstrated that a 256 bits priority encoder can achieve a time delay equal to 1.26 ns by using 2.5V 0.25 μ CMOS technology.

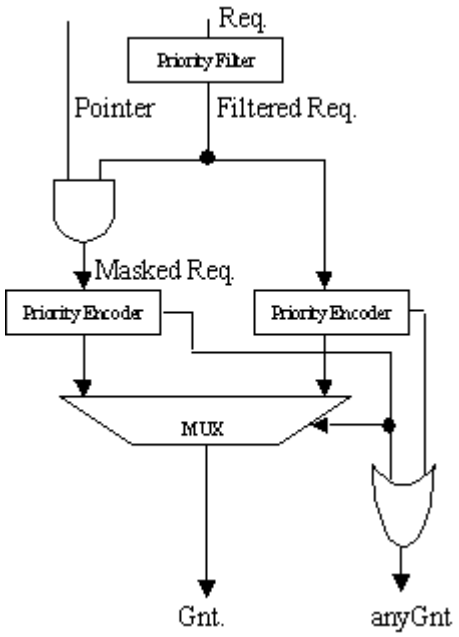

Figure 14. Arbiter hardware structure.

9. Generalization of DRDSRR

By exploiting the concept of randomized algorithms, actually not only RDSRR can be made to be stable, the other iterative scheduling algorithms may also achieve stability if similar modifications are employed. For example, using the iSLIP [5], we add one step to compute $S(t-1)$ and $R(t) = Z(t)$ at the beginning and after the iterative steps of iSLIP are executed, resulting in a matching $M'(t)$, then we add a step:

Accept. Let $M(t) = \arg \max_{S \in \{S(t-1), R(t), M'(t)\}} \langle S, Q(t) \rangle . M(t)$

is the present schedule. The corresponding inputs send accepts to the corresponding outputs.

In this case, the stability proof is similar to the proof of Lemma 3.

10. Conclusion

Maximum weight matching algorithms perform very well under non-uniform traffic, and are consequently stable. But they are too complex to implement. Their approximating algorithms are not much simpler too. Randomized algorithms are shown to achieve stability under any admissible traffic, but they inquire a high delay compared with iterative maximal matching algorithms (especially for somewhat uniform traffic). The group of SRR maximal matching algorithms have good delay performance, but are not stable under non-uniform traffic. In this paper, we have modified the RDSRR to derandomized RDSRR (DRDSRR) based on the concept of randomized algorithms. As a result, DRDSRR is shown to be stable under any admissible traffic while maintaining lower delay performance and is still simple to implement. We also make some improvement in hardware design. By making DRDSRR a maximal size matching, DRDSRR(v2) has improved performance over DRDSRR. We have also demonstrated that the hardware design of these algorithms is possible for switch sizes as large as 256x256 operating at OC-192 line rates. Finally, the basic idea of DRDSRR is shown that it can also be applied to other iterative scheduling algorithms so as to make them stable.

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] P. Gupta and N. McKeown, "Design and Implementation of a Fast Crossbar Scheduler," *IEEE Micro Magazine*, Jan-Feb 1999.