Distributed Load Adaptive Scheduling for High Speed Input Queued Switch

Shutao Sun
[1]Graduate School
Chinese Academy of Sciences
Beijing, China
Stsun@jdl.ac.cn

Youjian Zhao
Department of Computer Science and Technology, Tsinghua University
Beijing, China
zhaoyoujian@tsinghua.org.cn

Simin He, Yanfeng Zheng, Wen Gao[1,2]
[2]Institute of Computing Technology
Chinese Academy of Sciences
Beijing, China
{smhe, yfzheng, wgao }@jdl.ac.cn

Abstract—**Input queued switching architectures have become predominant in high speed switches and routers. In this paper, we change the point of view from weight-based matching to weight-based service, and propose a distributed load adaptive scheduling (DLAS) algorithm. In DLAS, the round robin arbiters are used to find a matching between the input ports and output ports. Once the matching between an input-output pair is established, the scheduler will keep it for a certain period which is a function of the number of cells queued in corresponding VOQ. Simulation results show that our scheme achieves high throughput and low delay under admissible traffic. For uniform Bernoulli i.i.d. traffic, it achieves 100% throughput, and for nonuniform traffic, its throughput is almost 100%.**

Keywords-switch; input queuing; scheduling; load adaptive

I. INTRODUCTION

Due to better scalability than output queuing, input queuing (IQ) [1], together with a crossbar switch fabric, virtual output queues (VOQs) [2] [6] and fixed-size cells, is now extensively used in high-speed routers. The scheduling of $N\times N$ input-queued routers in each time slot is a bipartite matching problem, and naturally many bipartite matching algorithms, maximum or maximal, weighted or not, are utilized in IQ scheduling.

By techniques of Lyapunov functions [3] and fluid models [4], it is proved that some maximum weighted matching (MWM) algorithms, e.g., LQF [6], guarantee 100% throughput asymptotically for admissible traffic patterns. However, such MWM algorithms have $O(N^3)$ or comparable complexity[8], too high to be used in on-line high-speed scheduling. iLQF, iOCF [6], iLPF [7], RPA [11] and MUCS [13] fall into maximal weighted matching. They have complexities ranging from $O(N^2)$ to $O(N^3)$. The complexity of the maximal weight matching algorithms also makes them no practical use since it prohibits the switch from scaling to large N or high line rate.

Practical designs pay more attention to maximal size matching algorithms, such as PIM [2], RRM, iSLIP [6], FIRM [9], DRRM [10], WFA and WWFA [11]. Most of them are based on parallel iterative matching, which iterates the following 3 steps:

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

Grant: If an unmatched output receives any requests, it chooses one of them to grant. The granted input is notified.

Accept: If the input receives at least one grant, it chooses one of them to accept.

Generally multiple iterations are needed to reach a maximal matching. Functional difference among the various algorithms exists only in the way that outputs choose which inputs to issue the grant to, and in the way inputs choose which grants to accept. This functional difference results in significant performance difference

PIM issues its grants and accepts by random selection. It is proved to converge in $O(\log N)$ iterations on average. Single iteration PIM can achieve a throughput of 63.2% under i.i.d. Bernoulli uniform traffic.

iSLIP replaces the random mechanism of PIM with round robin. By introducing smart round-robin pointer updating rules to desynchronize the pointers, iSLIP can achieve 100% throughput under uniform traffic even with single iteration.

DRRM adopts the unicast style to issue the requests, and it replaces iSLIP's three steps with only two. Due to the symmetry between inputs and outputs, iSLIP with single iteration and DRRM have similar performances.

All of PIM, iSLIP and DRRM use 1-bit information of each VOQ (empty or not) to make arbitration, which is incompetent for coping with non-uniform traffic patterns. By utilizing the multi-bit VOQ occupancy information to help arbitration, iLQF improves the performance of throughput under non-uniform traffic patterns. However, the multi-bit information exchange and comparison complicate the hardware design.

For a high speed switch, it is very difficult to accommodate multiple iterations in one time slot. SLIP (iSLIP with single iteration) and DRRM are reasonable candidates for practical application. However, their performances become poor under nonuniform traffic. EDRRM was proposed in [5] to improve the performance under burst and nonuniform traffic. In EDRRM, a VOQ will keep being served until it is exhausted; this service principle is too greedy and induces some demerits. First, it does not achieve 100% throughput under uniform traffic for a range of switch size. Secondly, it always amplifies the bursty of traffic, more serious under heavy load. Thirdly, its throughputs under some nonuniform traffic patterns are far from 100%.

In this paper, we aim at designing a scheduler for high performance switch. We wish our scheme to achieve all goals as follows: (1) Simple. Our algorithm should have the similar

This work was supported by the Knowledge Innovation Program in CAS and the National Natural Science Foundation of China under grant No.69983008.

complexity to maximal size matching. (2) Fast. Our algorithm should run at a quite high speed, and it had better find a good matching within only single iteration. (3) Effective. Our algorithm should achieve throughput near to 100%.

A distributed load adaptive scheduling (DLAS) algorithm is proposed to solve the problems of algorithms mentioned above. As we know, by finding a maximal weight matching in every time slot, algorithms, such as iLQF, achieve better performance than maximal size matching algorithms. In fact, maximal weight matching algorithms usually determine the weight according to the occupancy of VOQ or the port, and this leads the scheduling algorithms to favor the one with heavy load. While keeping the simplicity by using the round-robin arbiter, our algorithm achieves the same goal by providing service according to the number of cells queued in VOQ. Once an input and an output are matched, they will keep the connection for a reasonable period, which is a function of the number of cells queued in VOQ. Decisions on the length of match keeping period are based on local information, so it is easy to implement DLAS in parallel manner.

Although our scheme is very simple, simulation results show it is very efficient. Compared with DRRM, SLIP, and EDRRM, DLAS achieves higher throughput under nonuniform traffic. Unlike EDRRM, the throughput of DLAS isn't sensitive to the size of switch under uniform traffic, and asymptotic 100% throughput is achieved. Under the same traffic load, DLAS usually achieves smaller average cell delay.

This paper is organized as follows. In Section II we describe DLAS algorithm in detail. Then we evaluate the proposed algorithm by simulation in Section III. Some tricks for the implementation of DLAS are discussed in Section IV. At last, we end this paper with concluding remarks in Section V.

II. The Distributed Load Adaptive Scheduling Algorithm

Before we describe our scheduling algorithm, we introduce the concept of service function and discuss what feature it should own.

A. *To determine the service principles*

Our scheme serve a VOQ with consideration of the number of cells queued in it. The problem is: what is a good service function $s = S(x)$, where x is the number of cells queued in a VOQ, s is the number of cells transmitted continuously once the VOQ begins its transmission. We believe that for a given traffic pattern, there must be an optimal service discipline for it. However, since there are a large number of traffic patterns for a switch of a reasonable size, it is not easy to find a unified service discipline to achieve the optimal performance for all traffic patterns. So we must take a tradeoff. We try to find a service strategy that achieves good performance for most typical traffic patterns.

The occupancy of a VOQ is a suitable indicator of congestion, and the larger the number of cells queued in VOQ, the heavier the congestion. For an input queued switch with admissible traffic load, we argue that the congestion of a VOQ is the result of losing contention for the output ports or having less opportunity to participate in the previous contentions. Hence, once a congested VOQ gets an opportunity to transmit, the scheduler should compensate it to some extent. The more the queued cell in it, the more the scheduler should compensate it.

To find a good service function, we will choose from a set of candidates with two basic constraints as follows:

(1) $s=S(x)$ is monotonically increasing with x.
(2) $d=x-S(x)$ is monotonically increasing with x.

Constraint (1) keeps our scheme favoring the more congested VOQ, and constraint (2) tries to keep the burst in a reasonable range.

In this paper, we evaluate a set of service functions. We mainly pay attention to polynomial functions. Although logarithmic function $s = \log_2 x$ is easy to be implemented in hardware, it does not achieve performance good enough under nonuniform traffic. We exclude linear function because it is not sensitive enough when the number of queued cells is small and it increases too rapidly when the number of queued cells becomes large. Based on the simulation results, we give an advice on choosing a suitable service function.

B. *Algorithm*

In an $N \times N$ switch, at each input i, we maintain a separate queue VOQ_{ij} for each output j and a service counter c_i. For a matching between input i and output j, if counter $c_i > 0$, the matching will be kept. c_i decreases by one after a cell in VOQ_{ij} is transmitted. Once c_i decreases to zero, the matching is broken. We achieve above control function by dedicated handshake protocol and pointer updating rule. Similar to iSLIP, DLAS adopts round robin arbiter. All inputs and outputs are unmatched and the counters are set to 0 when a switch starts running. Then in each time slot, the following three steps are taken:

(1) Request: Each input i with $c_i > 0$ sends a request to the output corresponding to the current position of the pointer. Each input i with $c_i = 0$ sends a request to every output for which it has a queued cell, except the one to which input i sent a cell in the last time slot.
(2) Grant: If an output receives any request, it chooses the one that appears next in a fixed round-robin schedule starting from the current position of the pointer to grant and moves the pointer to the position corresponding to granted input if and only if the grant is accepted in Step (3). The output notifies each requesting input whether or not its request was granted. Otherwise the output grants the input corresponding to the current position of the pointer.
(3) Accept: If an input i receives a grant, it accepts the one that appears next in a fixed, round-robin schedule starting from the current position of the pointer. Assuming the accepted output is output j, if c_i=0, input i initializes c_i to $S(L(VOQ_{ij}))$, where $L(VOQ_{ij})$ is the number of cells queued in VOQ_{ij}, and moves the pointer of the input arbiter to the position corresponding to the accepted output. Then input i decreases c_i by 1, if c_i becomes 0, increase the pointer to one location beyond the accepted output.

Note that the grant/accept pointer stays at the position corresponding to granted/accepted input/output when counter $c > 0$. This will keep the matching between a matched pair if counter c of corresponding input is greater than 0. In Step (1), after finishing a continuous transmission (counter c becomes 0), the input pauses sending request to its just matched output for one time slot, and this gives its just matched output an opportunity to grant other input, hence terminates the matching between input and output. In Step (2), the output grants the input corresponding to the current position of the pointer if no request is sent to it. This mechanism can reestablish the just broken matching if the just matched input doesn't receive other grant except the one from its just matched output, even the input pauses sending a request to its just matched output.

III. PERFORMANCE EVALUATION

Since the performance of SLIP and DRRM are roughly comparable [14], we don't include the simulation result for DRRM in this paper. Before discussing our simulation results, we describe the traffic patterns used in our simulation.

A. Burstiness

Non-bursty (Bernoulli): In each time slot, a cell is generated by a source with some fixed probability, independent of all previous traffic.

Bursty: The burst traffic is generated as bursty arrival of cells (ON burst), followed by bursts of no cells (OFF burst). All cells within an ON burst have the same destination.

B. Load distribution

Uniform: The probability density function that determines the output is the same for all cells generated by the source. All input ports and output ports will handle the same traffic load.

Nonuniform pattern 1: In this traffic pattern, all the inputs have the same arrival rate. For each input i, 2/3 of arrivals are destined to output i, and the other arrivals are destined to output $(i + 1) \bmod N$.

Nonuniform pattern 2: In this pattern the arrival rate for each input is the same. From input i the traffic load to output $(i + j) \bmod N$ is two times the load to output $(i + k + 1) \bmod N$, $0 = k = (N-2)$. For example, assuming the arrival rate of an input is ?, the load at input 1 across outputs j is 2^{N-j}? / $(2^N - 1)$.

C. Simulation results

1) Throughput

a) Throughput under uniform traffic

Fig. 1 shows the throughput of EDRRM and DLAS with different service function under uniform Bernoulli i.i.d traffic. For the polynomial function we evaluate $s = \lfloor L^{1/3} \rfloor$, $s = \lfloor L^{1/2} \rfloor$, $s = \lfloor L^{3/4} \rfloor$, and $s = L$. DLAS–a denotes the DLAS with service function $s = \lfloor L^{\alpha} \rfloor$.

The results show that EDRRM is sensitive to switch size. Its throughput first decreases and then increases with switch size. For 6×6 switch, its throughput decreases to around 90%. However, our scheme outperforms EDRRM in throughput performance even if we choose anyone of the service functions mentioned above. Except the DLAS-1.0 and DLAS-0.75, the throughputs of DLAS all approach 100% (more than 99.9%). The throughput of DLAS-0.75 is also larger than 99.5%. We conjecture that for all $s = \lfloor L^{\alpha} \rfloor$, $0 = a < 1$, our scheme can achieve 100% throughput under uniform Bernoulli i.i.d traffic as long as the buffer is infinite. DLAS with appropriate service function is no longer sensitive to the switch size. This means our algorithm is more robust than EDRRM.

b) Throughput under nonuniform traffic

Figs. 2 and 3 present the simulation results under nonuniform i.i.d. traffic patterns 1 and 2. The performance of SLIP is quite poor. For DLAS, the throughput first increases and then decreases with a. We can see from these figures that when $a = 0.75$ or 0.5, the throughput is more than 95% for different switches. The throughputs of DLAS–0.75 and DLAS–0.5 are almost same for different switch size.

This result prompts us to simplify our evaluation by omitting the DLAS with poor service functions. Next, we mainly compare the performance of DLAS–0.5 and DLAS–0.75 with those of the existing algorithms.

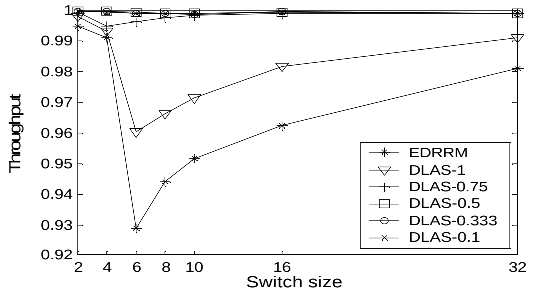

Fig.1. The throughput of DLAS and EDRRM under uniform traffic

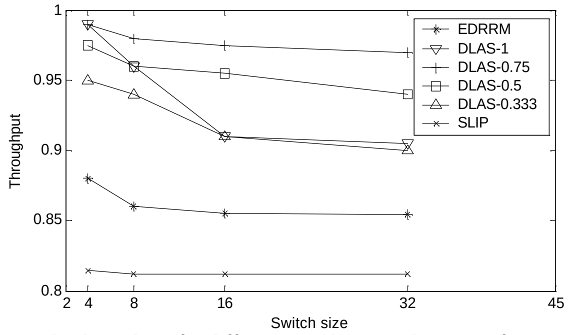

Fig. 2. The throughput for different switch size under nonuniform pattern 1

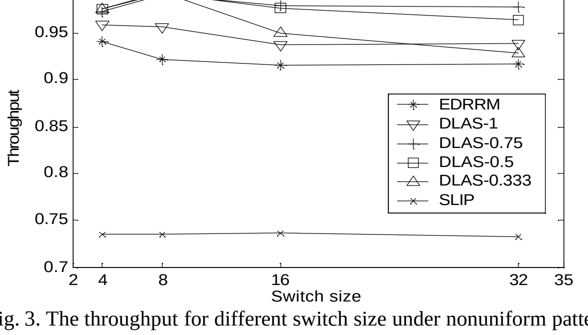

Fig. 3. The throughput for different switch size under nonuniform pattern 2

2) *Average delay*

a) Average delay under uniform traffic

Fig. 4 compares the delay performance of SLIP, EDRRM and DLAS under uniform i.i.d traffic for a 16 × 16 switch. The delays of DLAS–0.75 and DLAS–0.5 are comparable. Both of them outperform SLIP and EDRRM. The delay of EDRRM increases rapidly when the load is more than 60%.

Fig. 5 shows the average cell delay of DLAS–0.5, DLAS–0.75 and other algorithms under uniform burst traffic with average burst length of 16 cells for a 16 × 16 switch. The average cell delay for our algorithms is much smaller than those for other schemes.

b) Average delay under nonuniform traffic

Figs. 6 and 7 show the average cell delay under nonuniform patterns for a 16 × 16 switch. DLAS is well adaptive to different traffic models and achieves better performance.

3) *Burstiness at the output ports*

The burstiness of traffic has a negative effect on the successive hops on the transmission path. We evaluate the burstiness at the output ports after the traffic passes through a switch.

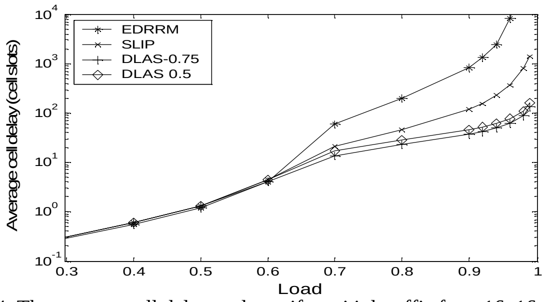

Fig. 4. The average cell delay under uniform i.i.d traffic for a 16x16 switch

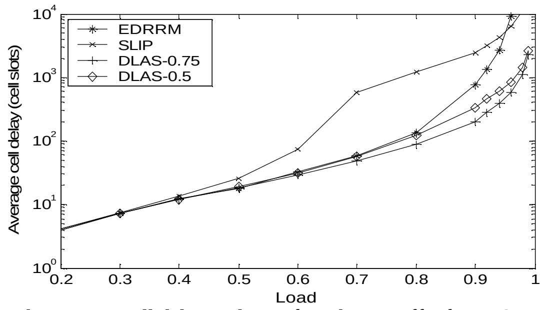

Fig. 5. The average cell delay under uniform burst traffic for a 16 × 16 switch

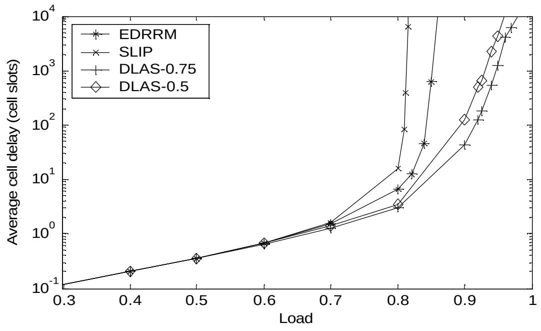

Fig. 6. Average cell delay under nonuniform pattern 1 for a 16 × 16 switch

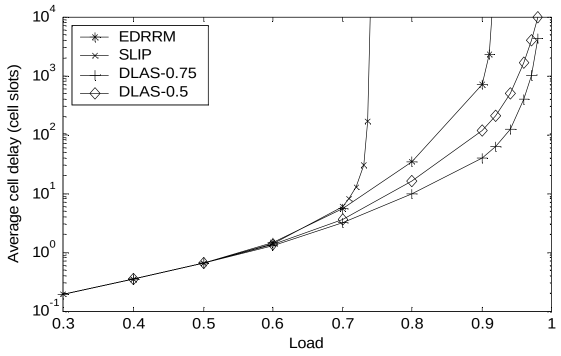

Fig. 7. Average cell delay under nonuniform pattern 2 for a 16 × 16 switch

a) Burstiness under uniform traffic

Fig. 8 shows the average bursty length of the traffic at the output ports of a 16 × 16 switch under uniform i.i.d traffic. The average bursty length is near to 1 for SLIP. For other algorithms, it increases with the traffic load. DLAS–0.75 and DLAS–0.5 usually keep the average bursty length less than 10. But EDRRM increases the average bursty length dramatically with the traffic load when the load becomes heavier. Fig. 9 shows the average bursty length of the traffic at the output ports under uniform burst traffic with average burst length of 16 cells. The average bursty length of output traffic for SLIP is monotonically decreasing with load. But for EDRRM, it is increasing. The DLAS–0.75 and DLAS–0.5 always keep burst length in a reasonable range.

b) Burstiness under nonuniform traffic

Fig. 10 and Fig. 11 show the average bursty length of the traffic at the output ports of a 16 × 16 switch under nonuniform Bernoulli traffic. The results show that under both nonuniform pattern 1 and nonuniform pattern 2, DLAS produces lower burst at the output ports.

From all our simulations under different traffic patterns, we can draw a conclusion that when 0.5 < *a* <0.75, the DLAS achieves satisfactory performance. Choosing the value of *a* is a tradeoff among the performances of burstiness, delay, throughput, and robustness. Our idea can easily be extended to other parallel iterative searching algorithms for input queued scheduling. Increasing the times of iteration will improve the performance. However, because of the limitation of space, we omit the evaluation of iterative DLAS.

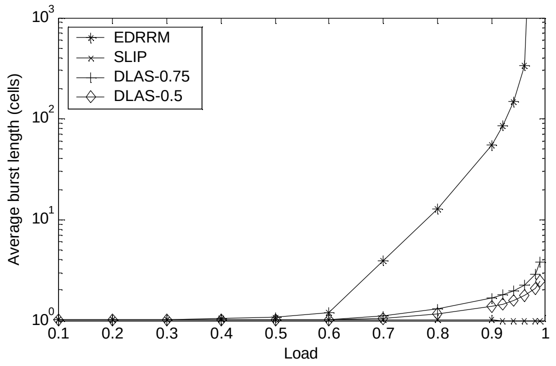

Fig. 8. The burstiness for a 16 × 16 switch under uniform i.i.d traffic

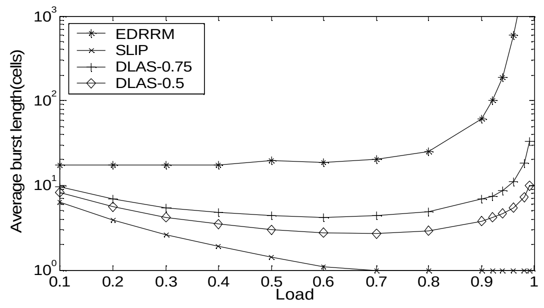

Fig. 9. The burstiness under uniform burst traffic for a 16×16 switch

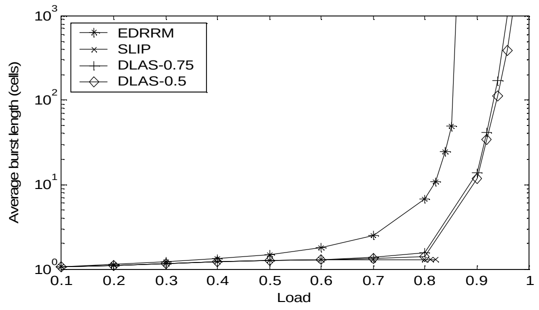

Fig. 10. The burstiness under nonuniform pattern 1 for a 16×16 switch

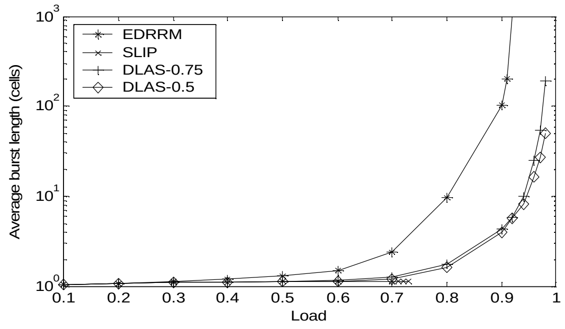

Fig. 11. The burstiness under nonuniform pattern 2 for a 16×16 switch

IV. TRICKS FOR IMPLEMENTATION

DLAS has the similar complexity to that of iSLIP. Calculation of service function can be implemented by fast table look-up. Simulation results show that DLAS can achieve satisfactory performance without producing large burst at output port. This implies that we can keep a table of small size.

The algorithm proposed in this paper assumes the scheduler can finish the calculation of service function within one time slot. However, it is not always the case. In order to solve this problem, we can make the scheduler work in a *Service* and *Break* mode. The *Service* and *Break* means that once input *i* and output *j* are matched, the input starts the transmission of the cells queued in VOQ$_{ij}$ and counts cells transmitted. At the same time the scheduler starts to calculate the service function. The transmission will be terminated if there is no cell in VOQ$_{ij}$ or the scheduler has finished the calculation of service function and finds the input has used up its quota. If there are few cells in VOQ, this usually means the switch is in the state of light load. It does not matter whether the calculation can be finished in time, and transmitting all queued cells is also OK. When load is heavy, the queue length becomes long, and there will be a few time slots for calculation. So DLAS can work correctly under heavy load and we get its gain.

In addition, we can improve the fairness by limiting the maximum value of counter c_i and/or initializing c_i to 1 instead of $S(L(VOQ_{ij})$ when input *i* and output *j* match again after they just finished a time of continuous transmission between themselves.

V. CONCLUSION

In this paper, we propose a distributed load adaptive scheduling algorithm for high performance switches. We transfer our attention from finding a maximum weight matching to serving the input ports in round-robin manner with a dynamic weight, which reflects the occupancy of VOQ. The complexity of our scheme is similar to that of iSLIP, and it is easy to be implemented in a distributed manner.

Simulation results show that if an appropriate service function is chosen, our scheme is robust and can achieve a throughput closing to 100% under admissible traffic even with single iteration. We propose DLAS–a with $0.5 = a = 0.75$ as a good candidate. Our scheme outperforms some other known algorithms in terms of delay-throughput performance under various traffic patterns tested in simulation. The features of our algorithm, fast, simple to implement, and of high performance, promise us that it is quite suitable for high performance switches.

REFERENCES

[1] M.J. Karol, M.G. Hluchyj, and S.P.Morgan. "Input versus output queuing on a space-division packet switch," *IEEE Trans. Commun.*, vol. 35, Dec. 1987, pp. 1347-1356.

[2] T. Anderson, S. Owicki, J. Saxe, and C. Thacker, "High speed switch scheduling for local area networks," *ACM Trans. Computer Systems*, vol.11, no. 4, Nov. 1993, pp. 319-352.

[3] N. McKeown, A. Mekkittikui, V. Anantharam, and J. Walrand, "Achieving 100% throughput in an input-queued switch," *IEEE Trans. Commun.*, vol. 47, Aug. 1999, pp. 1260-1272.

[4] JG. Dai and B. Prabhakar, "The throughput of data switches with and without speedup," in *Proc. INFOCOM*, Tel Aviv, 2000, pp. 556-564.

[5] Y. Li, S. Panwar, and H. J. Chao, "The dual round-robin matching switch with exhaustive service", in *Proc. HPSR 2002*, Kobe, Hyogo, May 2002, pp. 58-63.

[6] N. McKeown, "Scheduling algorithms for input-queued Switches," Ph.D. Thesis, University of California at Berkeley, 1995.

[7] A. Mekkittikui and N. MecKeown, "A practical scheduling algorithm to achieve 100% throughput in input-queued switches," in *Proc. INFOCOM*, San Francisco, 1998, pp.792-799.

[8] A. C. Kam and K. Siu, "Linear-complexity algorithms for QoS support in input-queued switches with no speedup," *IEEE Journal on selected areas in communications*, vol. 17, No.6, June 1999, pp. 1040-1056.

[9] D. N. Serpanos and P.I. Antoniadis, "FIRM: A class of distributed scheduling algorithms for high-speed ATM switches with multiple input queues," in *Proc. INFOCOM*, Tel Aviv, 2000. pp. 1634-1643.

[10] H. J. Chao and J.S. Pack, "Centralized contention resolution schemes for a large-capacity optical ATM switch," in *Proc. IEEE ATM Wksp.*, Fairfax, VA, May 1998, pp. 11-16.

[11] H. C. Chi and Y. Tamir, "Starvation prevention for arbiters of crossbars with multi-queue input buffers," in *Proc. COMPCON'94*, San Francisco, Feb. 1994, pp. 292-297.

[12] MA. Marsan, A. Bianco, E. Leonardi, and L. Milla, "RPA: A flexible scheduling algorithms for input buffered switches," *IEEE Trans. commun.*, vol. 47, no. 12, Dec. 1999, pp.1921-1933.

[13] H. Duan, J. W. Lockwood., S.M. Kang, and J. D. Will, "A high performance switching OC12/OC48 queue design prototype for input buffered ATM switches," in *Proc. INFOCOM*, Kobe, 1997, pp. 20-28.

[14] Y. Li, S. Panwar, and H. J. Chao, "On the performance of a Dual Round-Robin switch," in *Proc. INFORM*, vol. 3, April 2001, pp. 1688-1697.