An Efficient Scheduling Algorithm for Combined Input-Crosspoint-Queued (CICQ) Switches

Xiao Zhang and Laxmi N. Bhuyan
Department of Computer Science and Engineering
University of California, Riverside, CA 92521
Email: {xzhang, bhuyan}@cs.ucr.edu

***Abstract*— With today's ASIC technology, a large number of memory can be easily implemented in a single chip. This makes the combined input-crosspoint-queued (CICQ) crossbar switch a more attractive solution than the traditional input-queued (IQ) crossbar switch because of the simplicity of the CICQ switch scheduling. In this paper, we propose a shortest crosspoint buffer first (SCBF) scheme, and prove that it achieves 100% throughput for any admissible traffic. To facilitate hardware implementation, a maximal SCBF solution is also proposed. Our simulations show that the maximal SCBF performs almost identically to the maximum solution, and better than existing IQ and CICQ schemes. The time complexity of the maximal SCBF is $O(\log N)$, feasible for fast hardware implementation.**

I. Introduction

Crossbar is widely employed in current high-performance IP routers/switches [1], [2] because of its simplicity and non-blocking characteristics. However, the crossbar suffers from both input and output contentions: each input and output can only transfer one packet at a time. Output queuing (OQ) can avoid this situation. But to support OQ, the crossbar has to operate N times faster than the line card for an $N \times N$ switch. Therefore, input queuing is widely used in crossbar-based switches, and such a switch is called an input-queued (IQ) switch. To eliminate head-of-line (HOL) blocking [3], virtual output queuing (VOQ) [4] is used, where at each input, a logically separate FIFO queue is maintained for each output.

In an IQ switch, a switch scheduler is necessary. The difficulty of the switch scheduling comes from two aspects. First, the scheduling has to be performed in a very short time. Second, co-existence of input and output contentions make the switch scheduling more complicated than the OQ scheduling. The switch scheduler must make sure that matchings between inputs and outputs are conflict-free at any time. Theoretically, the maximum weight matching (MWM) algorithm [5][1] is proved to achieve 100% throughput for any admissible traffic. But its complexity is $O(N^3 \log N)$, too complicated to be implemented in hardware. Furthermore, multi-casting [6] and QoS guarantee [7] impose additional complexity on the switch scheduling.

This research has been supported by NSF grants CCR-0105676 and CCR-0311437.

[1]The MWM algorithm assigns each VOQ_{ij} a weight w_{ij}, and finds a matching M that maximizes $\sum_{(i,j)\in M} w_{ij}$. The weight can be queue length, waiting time or others.

Many algorithms [8]–[12] have been proposed to reduce the complexity. However, due to the time constraint, most of those proposed algorithms are still too complicated to be implemented in hardware. In practice, only simple algorithms, such as an iterative round-robin scheme iSLIP [12], can be deployed though performance is sacrificed.

Speedup is another approach to tackle the IQ switch scheduling problem. With a speedup of 2, Dai and Prabhakar proved that any maximal matching algorithms can achieve 100% throughput under any admissible traffic [13]; and Chuang et al. showed that an IQ switch can emulate an OQ switch [14]. However, speedup shortens the schedule time. Iterative-based algorithms may have no time to find a maximal matching.

Yet another approach is to add limited buffers inside the crossbar. This switch architecture has been investigated in many literatures [15]–[19] in which it is called a combined input-crosspoint-queued (CICQ) switch. With today's ASIC technology, a large amount of memory can be readily implemented in a single chip. This makes the CICQ switches more and more attractive. Stephens and Zhang applied distributed packet fair queuing to CICQ switches [20]. Magill et al. showed how to emulate an OQ switch by a CICQ switch with a speedup of 2 [21]. And Yoshigoe and Christensen did a survey on the evolution of CICQ switches [22].

One of the advantages of CICQ switches over IQ switches is the simplicity of the switch scheduling. Buffers inside the crossbar separate the input contentions from the output contentions, so that each input and output arbiter can make decisions independently. Algorithms such as OCF-OCF [15], RR-RR [16] have been proposed. LQF-RR [19] was also proved to achieve 100% throughput when the arrival rate of each input/output pair $\lambda_{ij} \leq 1/N$ for an $N \times N$ switch. However, to our best knowledge, no low cost algorithm has been proposed to provide 100% throughput for any admissible traffic without speedup.

We observe that output arbiters depend on the status of crosspoint buffers which in turn are determined by input arbiters. Therefore, input arbiters are more important than output arbiters in terms of throughput. In this paper, we propose a *shortest crosspoint buffer first* (SCBF) algorithm for input arbiters, and prove that *SCBF with any work-conserving output arbiters can achieve 100% throughput for any admissible traffic*.

The rest of the paper is organized as follows. In section II, we briefly describe the input queued and combined input-crosspoint-queued switch architectures. In section III, we propose the shortest crosspoint buffer first (SCBF) scheme, and provide the stability proof. In Section IV, we present simulation results to verify our analysis and compare with existing schemes. Finally section V presents our conclusions.

II. IQ SWITCHES VERSUS CICQ SWITCHES

A traditional $N \times N$ IQ crossbar switch, as shown in Fig. 1, operates on fixed-length units, called *cells*. A *slot* is the time to transfer a cell across the crossbar. Variable-length packets are segmented into cells on the input line card and re-assembled on the output line card. The limitation of an IQ crossbar switch is due to the co-existence of input and output contentions. In each slot, the crossbar scheduler has to configure the crossbar in such a way that one input is connected to at most one output and vice versa.

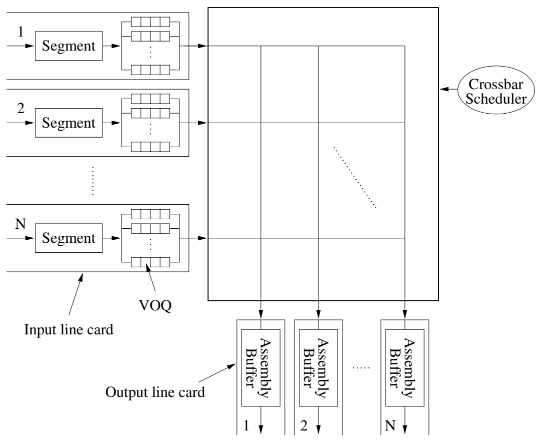

Fig. 1. An $N \times N$ IQ crossbar switch

An $N \times N$ CICQ switch, as shown in Fig.2, differs from an IQ switch by putting at each crosspoint a small buffer, called *crosspoint buffer* (CB). After a trivial re-arrangement of these buffers and connections, we have an equivalent graph shown in Fig. 3. Now, it is easy to see that an $N \times N$ CICQ switch in fact contains N $1 \times N$ crossbars (or de-multiplexers) between VOQs and CBs, and N $N \times 1$ crossbars (or multiplexers) between CBs and assembly buffers (ABs).

The key role of CBs is to separate the input contentions from the output contentions. This results in a two-stage scheduling scheme. In the first stage, each input arbiter determines which cell is transferred from a VOQ to the corresponding CB. When a CB is full, no more cells can be transferred to it. Note that if the CB size is unlimited (or large enough), this architecture is equivalent to output queuing and input arbiters are not necessary, because packets can directly go to CBs without buffering at VOQs. For a practical single-chip implementation, however, the CBs should be as small as possible.

In the second stage, each output arbiter selects cells from CBs to assembly buffers. It is exactly the same as the output-link scheduling. Any work-conserving scheduling algorithms

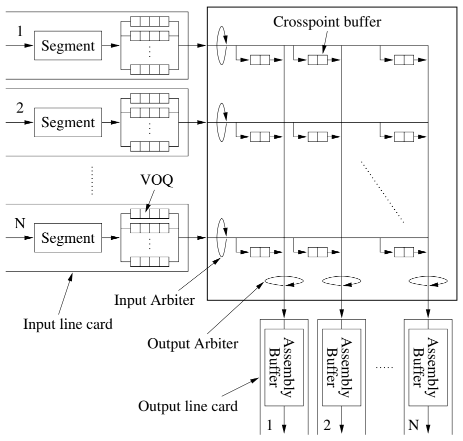

Fig. 2. An $N \times N$ CICQ crossbar switch

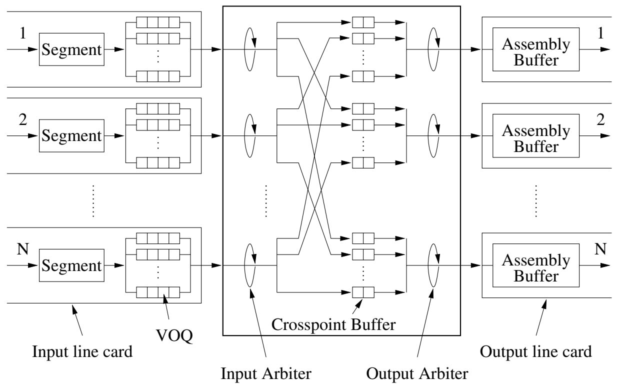

Fig. 3. An equivalent illustration of an $N \times N$ CICQ crossbar switch

suffice to give 100% throughput. Different algorithms differ only in packet delay.

However, because the output arbiters depend on the status of CBs which in turn are determined by the input arbiters. 100% thoughput may not be achieved if the input arbiters cannot delivery cells to CBs in time. The simple credit flow control mechanism used in existing schemes thus may fail to sustain full switch bandwidth.

In the next section, we describe an input arbitration scheme, called *shortest crosspoint buffer first* (SCBF) and prove that it can achieve 100% throughput for any admissible traffic with any work-conserving output arbitration schemes.

III. THE SCBF SCHEDULING ALGORITHM

A. Description of SCBF and its time complexity

In an $N \times N$ CICQ switch with each CB of K cells, let

- VOQ_{ij} be the virtual output queue for cells from $input_i$ to $output_j$.
- $V_{ij}(n)$ be the number of cells queued at VOQ_{ij} at the beginning of slot n.
- CB_{ij} be the crosspoint buffer for cells from $input_i$ to $output_j$.
- $C_{ij}(n)$ be the number of cells queued at CB_{ij} at the beginning of slot n, and

- $B_j(n) = \sum_i C_{ij}(n)$: the number of cells queued at all CBs destined to *output*$_j$ at the beginning of slot n.

In each slot n, there is an *edge* between *input*$_i$ and *output*$_j$ iff $V_{ij}(n) > 0$ and $C_{ij}(n) < K$, and the edge is assigned a *weight* of $B_j(n)$. The SCBF algorithm is to find a matching between inputs and outputs such that

1) each input with an edge is matched to an output with the smallest weight, so that the total weight of a matching is minimum.
2) In the case of multiple matchings having the same weight, choose the one with the maximum number of matched outputs.

The SCBF algorithm essentially favors the output with the least occupancy, so that each output is as work-conserving as possible. Since one-to-one matching is not required, the SCBF is much simpler than MWM. Fig. 4 shows the pseudo-code of the SCBF algorithm. The two sorts in line 1 and 2 take $O(N \log N)$. Finding an output (line 4) takes $O(N)$, and can be reduced to $O(\log N)$ using hardware parallel search circuit. Re-sorting outputs (line 6) also takes $O(\log N)$ because we just need to insert j into a sorted list. So the total time complexity of SCBF is $O(N \log N)$.

```
Initialization:
      $B_j(0) \leftarrow 0 \quad (j = 1, \cdots, N)$

In each slot $n$, SCBF works as follows:
1)   sort inputs in the increasing order of edge degree.
2)   sort outputs in the increasing order of $B$.
3)   for each sorted $input_i$ with an edge do
4)       find the first $j$ in the sorted output list such that
         there is an edge between $input_i$ and $output_j$.
5)       increase $B_j(n)$ by 1.
6)       re-sort outputs.

For each output_arbiter$_j$ working with SCBF, decrease
$B_j(n)$ by 1 if $B_j(n) > 0$.
```

Fig. 4. The SCBF Algorithm

B. Throughput analysis of SCBF

For the convenience of analysis, we assume that in each slot, a CICQ switch performs the following steps in sequence: 1) apply SCBF, 2) transfer cells from VOQs to CBs, 3) apply output arbitration, and 4) transfer cells from CBs to ABs.

We prove the stability of the SCBF algorithm using the fluid model technique [13]. Let $A_{ij}(n)$ be the number of packets that have arrived at *VOQ*$_{ij}$ up to time slot n, and $A_{ij}(0) = 0$. The arrival processes $\{A_{ij}(\cdot),\ i, j = 1, \ldots, N\}$ satisfy a strong law of large numbers (SLLN): with probability one,

$$\lim_{n\to\infty} \frac{A_{ij}(n)}{n} = \lambda_{ij} \quad i, j = 1, \ldots, N \tag{1}$$

where, λ_{ij} is the arrival rate at *VOQ*$_{ij}$. Let $D_{ij}(n)$ be the number of departures from *CB*$_{ij}(n)$ up to time slot n, and $D_{ij}(0) = 0$. A switch operating under a matching algorithm is said to be *rate stable* (equivalent to achieving up to 100% throughput), if, with probability one,

$$\lim_{n\to\infty} \frac{D_{ij}(n)}{n} = \lambda_{ij} \quad i, j = 1, \ldots, N \tag{2}$$

for any arrival processes satisfying (1). We also call a traffic *admissible* iff (1) holds and no inputs or outputs are oversubscribed, i.e.,

$$\sum_i \lambda_{ij} \le 1 \text{ and } \sum_j \lambda_{ij} \le 1 \tag{3}$$

Theorem 1: A CICQ switch operating under the SCBF algorithm with any work conserving output arbiters is rate stable under any admissible traffic that satisfies SLLN.

Proof: To prove that a switch is rate stable, it suffices to show that the corresponding fluid model is *weakly stable*, i.e., for every fluid model solution (D, T, Z) with $Z(0) = 0$, $Z(t) = 0$ for $t \ge 0$ (See Theorem 3 in [13] for proof).

Consider the fluid model of a CICQ switch operating under the SCBF algorithm. Let (D, T, Z) be a fluid model solution with $Z(0) = 0$, and $Z_{ij}(t)$ be the total amount of fluid queued at *VOQ*$_{ij}$ and *CB*$_{ij}$ at time t. From Lemma 1 in [13], to show that the fluid model is weakly stable, it suffices to show that $\dot{Z}(t) \le 0$ for any $Z(t) > 0$, where, for a function f, $\dot{f}(t)$ denotes the derivative of f at time t. Intuitively, we need to show that the rate of change of queue length is negative when there are backlogs in queues.

In each slot n, for any $Z_{ij}(n) > 0$, we have 4 cases:

1) $B_j(n) > 0$ and *output*$_j$ is matched by SCBF.
2) $B_j(n) > 0$ and *output*$_j$ is not matched.
3) $B_j(n) = 0$ and *output*$_j$ is matched by SCBF.
4) $B_j(n) = 0$ and *output*$_j$ is not matched.

- In case 1), 2) and 3), let $M_j(t) = \sum_{i'} Z_{i'j}(t)$ be the total amount of fluid destined for *output*$_j$ and queued at some VOQs and CBs at time t. $M_j(n+1) - M_j(n)$ is the difference in the number of arrivals at all inputs destined to $output_j$ at $n+1$ and the number of departures for $output_j$ at time n. The number of arrivals equals to $\sum_{i'}(A_{i'j}(n+1) - A_{i'j}(n))$. Because the output arbiters are work-conserving and either $B_j(n) > 0$ or *output*$_j$ is matched by SCBF, one cell must leave the switch for *output*$_j$ at the end of slot n. So we have

$$M_j(n+1) - M_j(n) \le \sum_{i'}(A_{i'j}(n+1) - A_{i'j}(n)) - 1 \tag{4}$$

Applying the fluid limit procedure and (3), we have

$$\dot{M}_j(t) = \sum_{i'} \dot{Z}_{i'j}(t) \le \sum_{i'} \lambda_{i'j} - 1 \le 0 \tag{5}$$

- In case 4), let $L_i(t) = \sum_{j'} Z_{ij'}(t)$ be the total amount of fluid queued at *input*$_i$ at time t. $L_i(n+1) - L_i(n)$ is the difference in the number of arrivals to $input_i$ at $n+1$ and the number of departures from $input_i$ at time n. The number of arrivals equals to $\sum_{j'}(A_{ij'}(n+1) -$

$A_{ij'}(n))$. Because $Z_{ij}(n) > 0$, $B_j(n) = 0$ and *output*$_j$ is not matched, *input*$_i$ must be matched to *output*$_{j'}$ such that $Z_{ij'}(n) > 0$, $j' \neq j$ and $B_{j'}(n) = 0$. We claim that j' is only matched to i, because if not, we can simply re-match i to j to increase the total number of matched outputs without increasing the total weight, and thus contradict to the fact that the SCBF algorithm gives the maximum number of matched outputs. Hence one cell must leave the switch from *input*$_i$ at the end of slot n. So we have

$$L_i(n+1) - L_i(n) \leq \sum_{j'} (A_{ij'}(n+1) - A_{ij'}(n)) - 1 \quad (6)$$

Applying the fluid limit procedure and (3), we have

$$\dot{L}_i(t) = \sum_{j'} \dot{Z}_{ij'}(t) \leq \sum_{j'} \lambda_{ij'} - 1 \leq 0 \quad (7)$$

Combine all cases, we have $\dot{Z}(t) = \sum_{ij} \dot{Z}_{ij}(t) \leq 0$ any for any $Z(t) > 0$. Therefore, *the fluid model is weakly stable, and hence the CICQ switch using SCBF is rate stable.* ■

C. A maximal approximation to SCBF

The above SCBF algorithm is a maximum solution, i.e., the matching decision is made globally. Each input needs to know the decisions of other inputs. For better scalability and faster hardware implementation, it is preferred that each arbiter makes decisions independently. This can be achieved by only requiring that each input with an edge is matched to an output with the smallest weight. The time complexity is therefore reduced to $O(\log N)$ with hardware parallel comparison circuit, which is clearly feasible for fast hardware implementation.

A matching obtained this way is maximal since the number of matched outputs may not be maximum, and equation (6) no longer holds. Fig. 5 illustrates the difference between the maximum and the maximal solution. When the maximum SCBF is used, all outputs are busy while in the case of the maximal SCBF, only output 0 is busy.

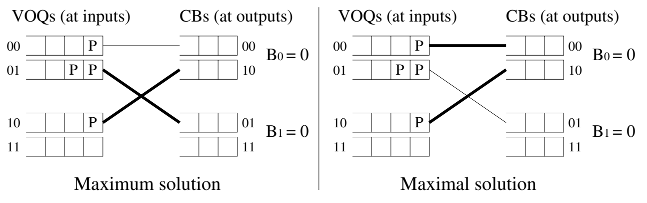

Fig. 5. Maximum SCBF vs. maximal SCBF (bold line is a matching)

However, in the worst case, if all inputs serve each output at the same time, the idle time of the last served output is at most N slots; and thereafter, all outputs will be busy. So we conjecture that the maximal SCBF can still achieve 100% throughput. Our extensive simulations show that in most cases, the difference between the maximal and the maximum SCBF is indistinguishable.

IV. Simulation Results

To evaluate the SCBF scheduling algorithm, we wrote a simulator to implement SCBF-RR and SCBF-OCF. We also implement iSLIP (with 3 iterations), MWM (using waiting time as weight), RR-RR, OCF-OCF, LQF-RR and output queuing for comparison. The switch size is 16×16. VOQs and output queues are statically partitioned with 64K bytes per input-output pair. Cell size is set to 64 bytes. Packet arrival is modeled as a 2-state ON-OFF process. The number of ON state slots is defined as the packet length which is obtained from a profile of NLANR trace at AIX site [23]. We collected 119,298,399 packets from May 12 02:45:06 2002 to May 18 23:11:34 2002. The packet length ranges from 20 to 1500 bytes with mean $E_{on} = 566$ bytes and standard deviation of 615 bytes. The number of OFF state slots is exponentially distributed with average $E_{off} = \frac{1-\rho}{\rho} E_{on}$, where ρ is defined as the offered workload ($0 < \rho < 1$). We evaluate our algorithms under various traffic patterns. Due to page limit, we only report some of the results.

We measure the delay of a packet from the first bit of the packet enters the VOQ to the last bit leaves the assembly buffer. Packets leave the assembly buffer in the FIFO order. Note that a packet is eligible to leave the assembly buffer when the last bit of the packet arrives at the assembly buffer. Measurement starts after a warm-up of 100,000 slots. Simulations run long enough to ensure the 95% confidence interval of the average delay with $\pm 5\%$ wdith.

A. Crosspoint buffer size

When arbitrations and cell transfers are performed sequentially in one slot, the CB size of 1 cell is enough to achieve maximum throughput. In practice (also in our simulation) pipeline is implemented such that each slot performs only one step, as illustrated in Fig 6. Since the input arbitration is independent from the output arbitration, in order to perform the input arbitration in slot 3, the minimum CB size of 3 cells is necessary to sustain the maximum throughput.

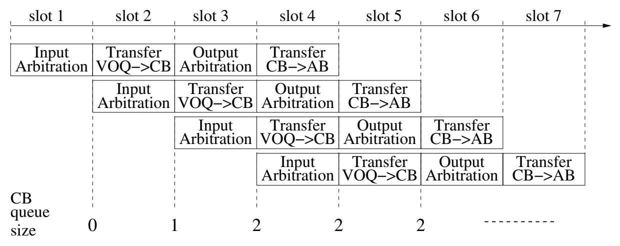

Fig. 6. Pipeline implementation of a CICQ switch. The minimum CB size of 3 cells is required to sustain the full switch bandwidth. Note that the pipeline can be shortened by combining step 2 and step 3, and the minimum CB size can be reduced to 2 cells.

B. Uniform Traffic

Fig. 7 shows the delay performance of various IQ/CICQ schemes under uniform traffic ($\lambda_{ij} = \rho/N \; \forall i,j$). As expected, all schemes perform well. At low workload, a CICQ switch has a slightly higher delay than an IQ switch due to the longer pipeline. When the workload is above 50%, SCBF-RR/OCF performs better than other schemes except output queuing.

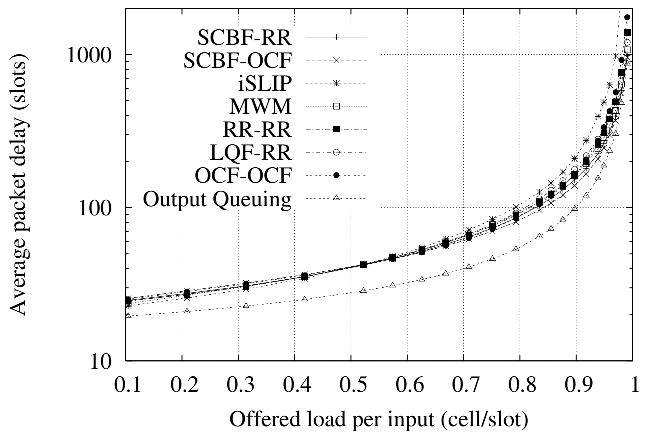

Fig. 7. Delay Performance of SCBF-RR/OCF under uniform traffic compared with other schemes. The crosspoint buffer size of CICQ switches is 8 cells.

Fig. 8 and Fig. 9 show the difference between the maximum and the maximal SCBF and the impact of different CB sizes on the delay performance. First, the maximum and maximal SCBF perform almost identically. Second, the CB size of 8 cells is enough to achieve good delay performance. This result in a total crosspoint buffer size of 128K bytes for a 16×16 switch or 512K bytes for a 32×32 switch, which is clearly implementable in today's ASIC technology.

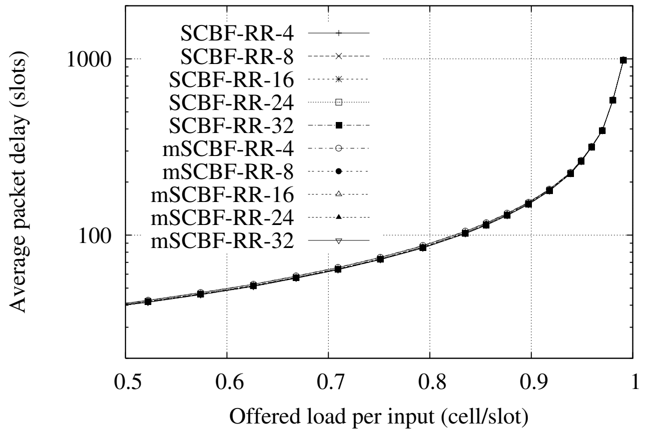

Fig. 8. Delay Performance of the maximum SCBF-RR and the maximal SCBF-RR (mSCBF-RR in the graph) with different CB sizes under uniform traffic.

C. Non-uniform Traffic

Next we report simulation results under the diagonal non-uniform traffic (used in [11]): $\lambda_{ii} = 2\rho/3$, $\lambda_{i|i+1|} = \rho/3$ and $\lambda_{ij} = 0$ for $j \neq i$ or $|i+1|$. This is a very skewed traffic pattern. Fig. 10 shows the average packet delay of traffic from *input*$_i$ to *output*$_i$ as a function of the workload per input. In this situation, iSLIP, RR-RR and OCF-OCF fail to achieve 100% throughput. Other schemes still perform well.

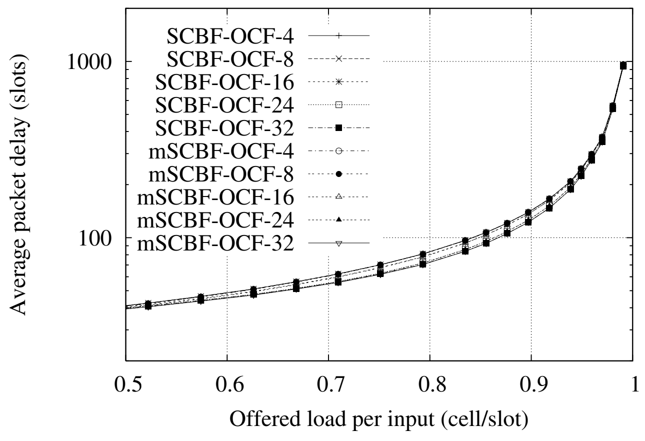

Fig. 9. Delay Performance of the maximum SCBF-OCF and the maximal SCBF-OCF with different CB sizes under uniform traffic.

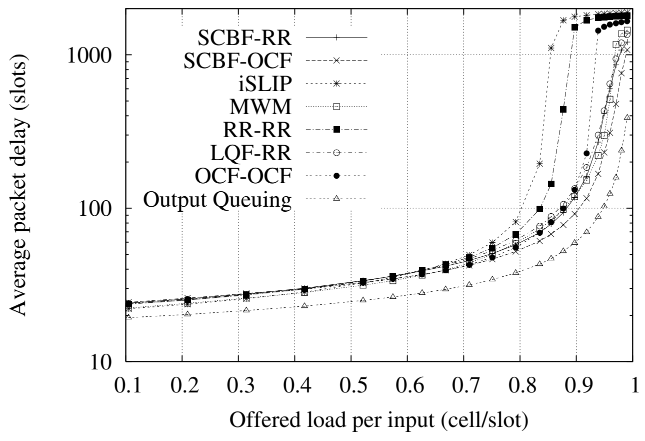

Fig. 10. Delay performance of SCBF-RR/OCF under diagonal traffic compared with other schemes. The CB size of CICQ switches is 8 cells.

We observe that all schemes except output-queuing experience packet loss under high workload. However, this does not invalidate the previous 100% throughput analysis because under very high workload, bursts of packets overload the outputs too often, and eventually overflow the VOQs. Fig. 11 shows the packet loss rate of different schemes. At a workload of 94-97%, SCBF-RR incurs slightly higher loss rate (hence higher packet delay) than MWM. When the workload reaches 98%, SCBF-RR has lower loss rate. SCBF-OCF, on the other hand, always has the lowest loss rate, thus the lowest packet delay.

Fig. 12 and Fig. 13 compare the maximum and the maximal SCBF with different CB sizes. When the CB size is 4 cells, the maximal SCBF performs worse than the maximum one. When the CB size reaches 8 cells, the difference becomes indistinguishable.

V. Conclusion

In this paper, we proposed a novel shortest crosspoint buffer first (SCBF) algorithm for input arbiters of a CICQ switch, and proved that a CICQ switch operating under SCBF with any

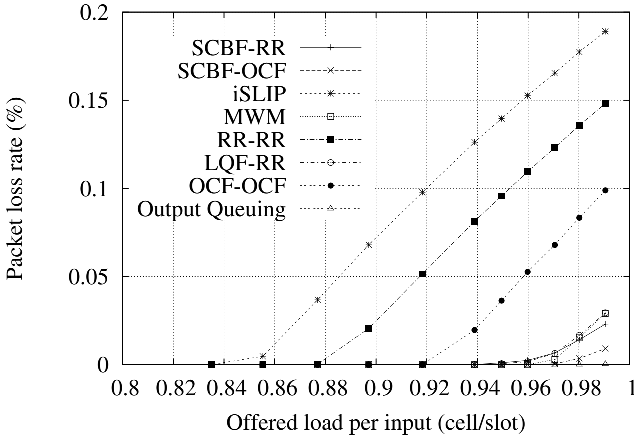

Fig. 11. Packet loss rate of different schemes under diagonal traffic. The CB size of CICQ switches is 8 cells.

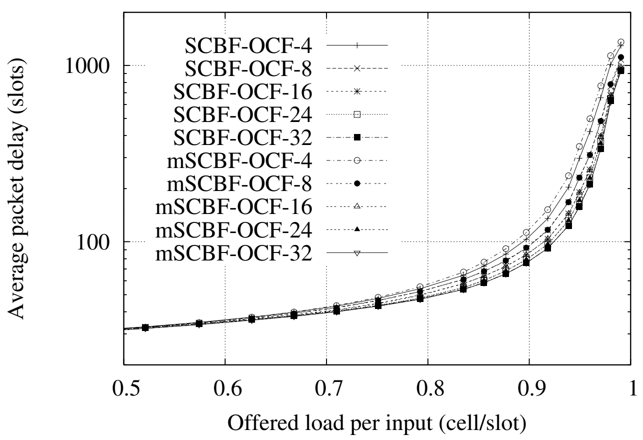

Fig. 13. Delay Performance of the maximum SCBF-OCF and the maximal SCBF-OCF with different CB sizes under diagonal traffic.

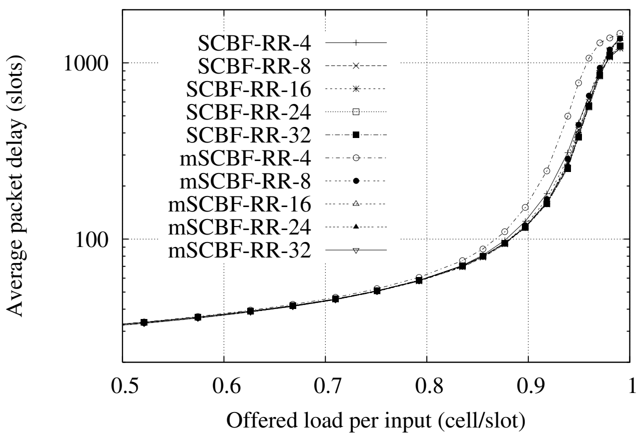

Fig. 12. Delay Performance of the maximum SCBF-RR and the maximal SCBF-RR with different CB sizes under diagonal traffic.

work-conserving output arbiters can achieve 100% throughput for any admissible traffic. Our simulations under uniform and non-uniform traffic showed that SCBF achieves lower delay and packet loss rate than existing IQ/CICQ schemes. In addition, the maximal SCBF performs almost identically to the maximum solution. Its $O(\log N)$ time complexity indicates that it is feasible for high performance switches.

References

[1] Cisco Systems, Inc, "Cisco 12000 series internet routers." [Online]. Available: http://www.cisco.com

[2] C. Partridge, *et al.*, "A 50-Gb/s IP router," *IEEE/ACM Trans. Networking*, vol. 6, no. 3, pp. 237–248, June 1998.

[3] M. J. Karol, M. G. Hluchyj, and S. P. Morgan, "Input versus output queuing on a space-division packet switch," *IEEE Trans. Commun.*, vol. COM-35, no. 12, pp. 1347–1356, Dec. 1987.

[4] Y. Tamir and G. Frazier, "High performance multi-queue buffers for VLSI communication switches," in *Proc. 15th Ann. Symp. Computer Architecture*, June 1988, pp. 343–354.

[5] N. McKeown, V. Anantharam, and J. Walrand, "Achieving 100% throughput in an input-queued switch," in *Proc. IEEE INFOCOM'96*, vol. 1, Mar. 1996, pp. 296–302.

[6] B. Prabhakar, N. McKeown, and R. Ahuja, "Multicast scheduling for input-queued switches," *IEEE J. Select. Areas Commun.*, vol. 15, no. 5, pp. 855–866, 1997.

[7] C.-S. Chang, W.-J. Chen, and H.-Y. Huang, "On service guarantees for input buffered crossbar switches: a capacity decomposition approach by birkhoff and von neumann," in *IEEE IWQoS'99*, London, U.K., 1999, pp. 79–86.

[8] Y. Tamir and H. C. Chi, "Symmetric crossbar arbiters for VLSI communication switches," *IEEE Trans. Parallel Distrib. Syst.*, vol. 4, pp. 13–27, Jan. 1993.

[9] T. E. Anderson, S. S. Owicki, J. B. Saxe, and C. P. Thacker, "High speed switch scheduling for local area networks," *ACM Transactions on Computer Systems*, vol. 11, no. 4, pp. 319–352, Nov. 1993.

[10] A. Mekkittikul and N. McKeown, "A practical scheduling algorithm to achieve 100% throughput in input-queued switches," in *Proc. IEEE INFOCOM'98*, vol. 2, Apr. 1998, pp. 792–799.

[11] P. Giaccone, D. Shah, and B. Prabhakar, "An implementable parallel scheduler for input-queued switches," *IEEE Micro*, Jan./Feb. 1999.

[12] N. McKeown, "The iSLIP scheduling algorithm for input-queued switches," *IEEE/ACM Trans. Networking*, vol. 7, no. 2, pp. 188–201, Apr. 1999.

[13] J. G. Dai and B. Prabhakar, "The throughput of data switches with and without speedup," in *Proc. IEEE INFOCOM'00*, vol. 3, Mar. 2000, pp. 556–564.

[14] S.-T. Chuang, A. Goel, N. McKeown, and B. Prabhakar, "Matching output queueing with a combined input/output-queued switch," *IEEE J. Select. Areas Commun.*, vol. 17, no. 6, pp. 1030–1039, June 1999.

[15] M. Nabeshima, "Performance evaluation of a combined input- and crosspoint-queued switch," *IEICE Trans. Commun.*, vol. E83-B, no. 3, pp. 737–741, Mar. 2000.

[16] R. Rojas-Cessa, E. Oki, Z. Jing, and H. J. Chao, "CIXB-1: combined input-one-cell-crosspoint buffered switch," in *Proc. IEEE HPSR'01*, May 2001, pp. 324–329.

[17] R. Rojas-Cessa, E. Oki, and H. J. Chao, "CIXOB-k: combined input-crosspoint-output buffered packet switch," in *Proc. IEEE GLOBECOM'01*, Dec. 2001, pp. 2654–2660.

[18] K. Yoshigoe and K. J. Christensen, "A parallel-polled virtual output queued switch with a buffered crossbar," in *Proc. IEEE HPSR'01*, May 2001, pp. 271–275.

[19] T. Javidi, R. Magill, and T. Hrabik, "A high-throughput scheduling algorithm for a buffered crossbar switch fabric," in *Proc. IEEE ICC'01*, June 2001, pp. 1581–1587.

[20] D. C. Stephens and H. Zhang, "Implementing distributed packet fair queuing in a scalable switch architecture," in *Proc. IEEE INFOCOM'98*, vol. 1, Mar. 1998, pp. 282–290.

[21] R. B. Magill, C. E. Rohrs, and R. L. Stevenson, "Output-queued switch emulation by fabrics with limited memory," *IEEE J. Select. Areas Commun.*, vol. 21, no. 4, May 2003.

[22] K. Yoshigoe and K. J. Christensen, "An evolution to crossbar switches with virtual output queuing and buffered cross points," *IEEE Network*, vol. 17, no. 5, Sept./Oct. 2003.

[23] National Laboratory for Applied Network Research, "NLANR network traffic packet header traces." [Online]. Available: http://pma.nlanr.net/Traces/Traces